NTNU

Norwegian University of
Science and Technology

Computational Linguistic Creativity: Poetry generation given visual input

Malte Loller-Andersen

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Björn Gambäck, IDI

Norwegian University of Science and Technology
Department of Computer Science

Malte Loller-Andersen

Computational Linguistic Creativity: Poetry generation given visual input

Master's Thesis in Computer Science, Spring 2017

Data and Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Abstract

Poetry is one of the most complex Natural Language Generation (NLG) tasks, because the value of poetry is very dependent of both the form and content. This Master's Thesis describes the implementation and evaluation of a system able to generate poetry satisfying rhythmical and rhyming constraints from an input image.

In the last few years, deep learning has been introduced into the field of poetry generation, which has made good use of these Artificial Neural Networks. In the field of object recognition, Convolutional Neural Networks are showing the best results, and are most widely used.

The poetry generation system consists of a Convolutional Neural Network for image object classification, a module for finding related words and rhyme words, and a Long Short-Term Memory (LSTM) Neural Network trained on a song lyrics data set compiled specifically for this Thesis.

In total, 153 stanzas were generated and evaluated in two different experiments. The results of these experiments indicate that the deep learning based system is capable of generating subjectively poetic, grammatically correct and meaningful poetry, but not on a consistent basis.

Sammendrag

Poesi er en av de mest interessante og komplekse naturlig språkgenererings utfordringer, fordi verdien av poesi er basert på både form og innhold. Dette gjør poesigenerering til et godt felt å forske på.

For å få en bedre forståelse av fagfeltene innenfor poesigenerering og objekt klassifisering av bilder, ble state-of-the-art datamaskinsystemer for begge disse feltene utforsket. Dyp læring har de siste årene blitt introdusert i begge fagfeltene, hvor forskjellige typer av kunstige neurale nettverk ofte er brukt.

Denne masteroppgaven beskriver et design, implementasjon og evaluering av et datamaskinsystem som er i stand til å generere poesi gitt poetiske restriksjoner, samt et input bilde. Systemet består av et konvolusjonelt neuralt nettverk for å klassifisere input bildet, en modul for å finne relaterte termer og rimord, og et Long Short-Term Memory (LSTM) nettverk trent på sangtekster som er samlet sammen i forbindelse med denne masteroppgaven.

Datamaskinsystemet genererte totalt 153 dikt og evaluerte diktene i to forskjellige eksperimenter. Resultatet av disse eksperimentene indikerer at systemet er i stand til å generere subjektivt poetisk, grammatisk korrekt og meningsfull poesi.

Preface

This Master's Thesis is submitted to the Norwegian University of Science and Technology (NTNU) as a part of the requirements for the degree of Master of Science in Computer Science. The Master's Thesis work has been performed at the Department of Computer Science, NTNU, Trondheim. The Thesis was supervised by Professor Björn Gambäck.

I would like to thank my supervisor Björn Gambäck for his guidance, feedback and help throughout this Master's Thesis, and Simen Selseng as we attended meetings together, and for many interesting discussions. I would also like to thank all the people who participated in the experiments that were conducted.

Malte Loller-Andersen
Trondheim, 11th June 2017

Contents

1. Introduction **1**
- 1.1. Project Goals . . . 1
- 1.2. Contributions . . . 3
- 1.3. Thesis Structure . . . 3

2. Background Theory **5**
- 2.1. Poetry . . . 5
  - 2.1.1. Poetry definition . . . 5
  - 2.1.2. Rhyme and rhyme schemes . . . 6
  - 2.1.3. Rhythm, Syllables and Stress Patterns . . . 8
- 2.2. Natural Language Processing . . . 9
- 2.3. Computational algorithms . . . 10
  - 2.3.1. Deep machine learning . . . 10
  - 2.3.2. Artificial Neural Networks . . . 11
- 2.4. Evaluation metrics of poetry . . . 20
- 2.5. Frameworks . . . 22
  - 2.5.1. ConceptNet . . . 22
  - 2.5.2. TensorFlow . . . 23
  - 2.5.3. Natural Language ToolKit . . . 24

3. Related Work **27**
- 3.1. State-of-the-art poetry generation . . . 27
  - 3.1.1. Template Based Poetry Generation . . . 27
  - 3.1.2. Generate and Test Approaches . . . 28
  - 3.1.3. Evolutionary Algorithms . . . 29
  - 3.1.4. Case-Based Reasoning (CBR) Approaches . . . 31
  - 3.1.5. Corpus-based Methods . . . 32
  - 3.1.6. Deep Learning Methods . . . 33
- 3.2. State-of-the-art image classification . . . 36

4. Data set **43**
- 4.1. Gathering the data set . . . 43
- 4.2. Preprocessing of the data . . . 44
- 4.3. Final data sets . . . 46

5. Architecture **49**
5.1. Inception 49
5.2. Finding related words and rhyme pairs 51
5.3. The Long Short-Term Memory (LSTM) network 52
5.4. Poetry generation process 56

6. Experiments and Results **59**
6.1. Training of the LSTM network 59
6.2. Experiments on the generated poetry 61
6.2.1. Grammaticality, poeticness and meaningfulness 62
6.2.2. Human- and machine generated poetry experiment . . 69
6.2.3. Participant Sentiments and Qualitative Observations . 72

7. Evaluation and Conclusion **73**
7.1. Evaluation 73
7.2. Discussion 75
7.3. Conclusion 76
7.4. Possible future work 77

Bibliography **79**

Appendix **85**
A. Various poems generated by the system 85

List of Figures

2.1. The most basic form of the perceptron 12
2.2. A Feed Forward Neural Network 14
2.3. A mapping from the input layer to a convolutional layer . . . 16
2.4. An example of the pooling layer 17
2.5. The core idea behind a RNN 18
2.6. A Long Short-Term Memory chain 19

4.1. The tree structure of *www.mldb.org* 44

5.1. A high level overview of the system 50
5.2. Architecture of the LSTM network 54
5.3. The search tree of the system 56

6.1. A randomly chosen selection of the poems generated 67
6.3. The setup for Experiment 2 71

A.1. Ten of the highest rated poems. 89
A.2. Ten of the lowest rated poems. 94

List of Tables

2.1. Three different activation functions 13
2.2. Inception-v3 performance 24

4.1. The process of choosing the vocabulary size 46
4.2. Data sets properties. 46

6.1. Word perplexity for different architectures 60
6.2. Word appearances in different data sets 61
6.3. AVG, SD and MD of poems evaluated 62
6.4. Comparisons between poems 68
6.5. Correlations between poems 69
6.6. Results of the second experiment 71

7.1. Percentage of poems containing I's 74

1. Introduction

The field of Computational Creativity is a sub-field of Artificial Intelligence in the cross-section of Cognitive Science containing elements from philosophy, linguistics and arts. The goal of computational creativity is to create computer systems that create new content, let it be musical pieces, text or art. Wiggins (2006) defines computational creativity as the following:

> *The performance of tasks (by a computer) which, if performed by a human, would be deemed creative*

This is a fairly loose description, but a description nonetheless. Having computers develop creative content can help humanity understand how their creativity works by a way of algorithms, and maybe look into new areas regarding human-thinking creativity.

Computational linguistic creativity involves theoretical study of language as well as developing computer algorithms to process language. It uses a varied arsenal of machine learning, artificial intelligence and statistics to generate, predict and extract the meaning of text. This includes story narratives, jokes, analogies, word associations, poetry, and much more. In Natural Language Generation (NLG), poetry is one of the more interesting and complex challenges, because its value depends on both the form and its content. Therefore, it makes a good field to research.

In this Thesis, a system capable of taking images as input and return a short stanza is designed and implemented. A state-of-the-art poetry generator is implemented to generate poetry while using state-of-the-art image classification to find objects in an image. The image classifier is a Convolutional Neural Network (CNN) already pre-trained, called Inception (Szegedy et al., 2016). The poetry generator combines tree search with a Long-Short Term Memory (LSTM) network trained on a custom made data set.

1.1. Project Goals

In this section the main goals for this project are presented.

G1: Investigate computational poetry generation and image classification

In order to properly develop an algorithm to create poetry, one need to look into what the state-of-the-art poetry generators are composed of. Furthermore, to be able to recognize objects in images, the field of image classification also needs to be researched.

G2: Develop a data set from scratch

A data set is necessary to train the machine learning algorithm used in this project, and none are available online. The data set has to be gathered and properly pre-processed.

G3: Evaluate the results of a system that generates poetry from visual input

Design, develop and implement a system that can take an image as visual input and generate a topical poem of the objects found in the image. Evaluate these results.

RQ1: What are the effects of using song lyrics when training an LSTM network, on the generated poetry?

A data set is needed for training the model, and using song lyrics instead of other poetry as a foundation, can reveal interesting properties.

RQ2: Are the predictions from an LSTM-network enough to achieve grammatically correct poetry?

The implications of this question are that poeticness and meaningfulness will be addressed by further help from other modules than the LSTM network. Grammaticality, however, will mostly only be determined by predictions from the network.

RQ3: In what ways can the generated poetry be evaluated?

The generated poetry from this system is evaluated on Manurung (2004)'s three properties: Grammaticality, poeticness and meaningfulness. These terms are described in depth in Section 2.4.

1.2. Contributions

The main contributions from this Master's Thesis are:

1. A literary study of the state-of-the-art poetry generation and image object classification.
2. A data set built from scratch from more than 200,000 songs.
3. Design and implementation of a system able to create poetry from an input image.
4. Instead of using rule-based and fill-in methods, this system actively predicts outcomes in a more creative way. By combining tree search with deep learning, searching for optimal paths with suitable rhyme words, it guarantees strophically correct poems.

1.3. Thesis Structure

In Chapter 2 the background material, such as the poetry definition and neural networks, needed to understand this Thesis is introduced. It also mentions the frameworks and resources used, as well as different ways to evaluate poetry.

Chapter 3 describes the state-of-the-art in both poetry generation and image classification. The data set gathering of more than 200,000 songs and the pre-processing of them are explained in Chapter 4.

Chapter 5 talks about the architecture of the whole system implemented to generate poetry, and Chapter 6 shows the results gathered from the system.

In Chapter 7 a discussion and evaluation of the degree to which the project goals have been achieved is presented. In addition, the conclusions are drawn and possible future work is discussed.

2. Background Theory

This chapter covers useful information about poetry helping to understand the poetry generation system. It also contains different ways of measuring the system based on the poetry it generates, as well as giving a brief overview of deep learning and specifically neural networks. Additionally, it covers various frameworks and resources used in this project.

2.1. Poetry

Linguistic theory has been well defined and formalized over the past several hundred years and is still changing today. New words are being created, while old words are changed and forgotten. This section will define poetry and cover relevant information for this Thesis. First a definition of poetry and what kind of poetry is generated is introduced. Then a brief introduction to rhymes is given, followed by an introduction of rhythm, syllables and stress patterns.

2.1.1. Poetry definition

Poetry has various different forms and is a very subjective genre, therefore a concrete definition of poetry can be hard to nail down. The Merriam-Webster English Dictionary defines poetry as:

> *writing that formulates a concentrated imaginative awareness of experience in language chosen and arranged to create a specific emotional response through meaning, sound, and rhythm*

This definition shows that both content and form have a big influence on the value of poetry. This is what differentiates poetry from prose text in the first place. When translating poetry, a lot of the meaning, rhyming and rhythm is lost. It is because of the loss of rhymes and rhythm that the poetry loses value. In a prose text, however, if rhythm and rhymes disappear, the prose text does not necessarily lose any value. Levin (1962) also supports this:

> *... in poetry the form of the discourse and its meaning are fused into a higher unity ... form in fact embraces and penetrates message in a way that constitutes a deeper and more substantial meaning than either abstract message or separable ornament.*

With these broad definitions, it is hard to draw any conclusions about the poetry generated in this Thesis. Therefore, the definition of poetry used in the rest of this report is the following:

> *A set of lines satisfying rhyming and rhythmic constraints. The text generated consists of one stanza containing Y lines. The lines have to be a length of X syllables, and a line must rhyme with another line.*

By using this definition, it is clear what kind of poetry is being generated, and conclusions can be based on this. Furthermore, a stanza is a group of lines separated from others in a poem, often to shift between action, moods and thoughts. The terms Stanza and poetry will be used interchangeably about the poetry generated by this system.

2.1.2. Rhyme and rhyme schemes

Rhyming is the repetition of similar sounds in two or more words. More specifically, Whitfield (1951) formally defines rhyme as

> *Rhyme is the correspondence, in two or more words or verses, of terminal sounds beginning with an accented vowel, which, in modern English usage, must be preceded by different consonant sounds, or by a consonant in one case and none in the other.*

Furthermore, *Whitfield* writes that there are two rules in rhyming:

- The rhyme sounds must be identical.
- Each rhyme used must vary from all others in the consonant-sounds that precede the rhyme.

Examples of these are words like *bright* and *light*. Another example is *fair* and *care*. Notice in the second example that the ending of the words are written differently (*air* and *are*), which is possible because it is the pronunciation of the words which is in focus, not how they are written. For instance, words like *care* and *are* have the same ending, but do not rhyme.

The rhymes described above are called perfect rhymes. They both share a common accented vowel and consonant in the terminal of the word. *Imperfect rhymes*, also called *slant rhymes* are words that only share a terminal vowel, e.g. *heart* and *star* or share a terminal consonant such as *talk* and *milk*.

Other types of rhymes include *Rich rhymes* and *Eye rhymes*. *Rich rhymes* use two different words that sound the some (i.e. homonyms), for example *raize* and *raise*. *Eye rhymes* rhyme on words that look the same, but are pronounced differently. One example is *rough* and *bough*. In this Thesis, perfect rhymes will be favored over other forms of rhymes.

Furthermore, a distinction is made between **masculine** and **feminine** rhyme. In masculine rhymes, only one syllable rhymes, e.g. *mean* and *seen*, and in feminine rhymes, where the rhyme occurs over two or more syllables, e.g. *lighting* and *fighting*.

Another side to rhyming is *rhyming schemes*. These describe the patterns of rhyme that occur in the poem. The most common rhyme scheme is the *ABAB* rhyming scheme, also called alternate rhyme. In the *ABAB* rhyme scheme, each of the four letters represents a line. If the first line has an A rhyme, and the second line has a B rhyme, then the third line also has an A rhyme and the fourth line has a B rhyme. A quick example to showcase this is

... clear
... dream
... here
... cream

Other rhyme schemes include the *AABB* rhyme scheme, *ABBA*, and so on. The finished system supports any combination of rhyme schemes.

According to Attridge and Roberts (1987) it is rhyme, described above, and these three definitions that make up the various phonemic patterns:

- **Alliteration**: Refers to the repetition of a word's opening sound such as *butt* and *bend*.

- **Assonance** is the repetition of identical vowel sounds in different words, that appear close to each other. It is also called *half rhyme*. E.g. *deep* and *green*.

- **Consonance** is when the ending consonant between two words are the same: e.g. *bed*, *bad*.

2.1.3. Rhythm, Syllables and Stress Patterns

Syllables and stress patterns of words and sentences are very important concepts to nail down before creating poetry because of the flow and rhythm of your poem.

A syllable is a unit of organization for a sequence of speech sounds. For example the word *teddybear* is composed of three syllables: *te* - *ddy* - *bear*. One syllable is made of a syllable *nucleus* (in English this is often a vowel), an optional start, called the *onset*, and an ending called the *coda*. The onset and coda are often consonants in the English language. If a word is made of one syllable it is called *monosyllabic*. Two-syllable words are called *disyllabic*, while *trisyllabic* is used for words with three syllables. Finally, if a word consists of more than three syllables it is called *polysyllabic*.

The term *stress* is used here to refer to an abstract property of syllables within the domain of a word. A syllable can have three different types of stress: unstressed, primary stressed and secondary stressed. When a word has more than one syllable, a single syllable within the word is often given more emphasis than the other syllables. That syllable is called the stressed syllable. The stressed syllable is pronounced louder, longer and often with a higher pitch than other syllables. An unstressed syllable is often located next to a stressed syllable. This vowel is not phonetic, but instead is pronounced with a quick, neutral sound. The secondary stressed syllable most often occurs two beats off a primary stressed syllable. It is given more emphasis than an unstressed syllable, but less emphasis than a stressed one. For this project, primary and secondary stressed syllables are merged, thus only stressed and unstressed syllables are present.

The rhythmic pattern of a verse, represented by these stressed and unstressed syllables is called a *metre*. Four different kinds of metre are identified:

- Strong stress metre
- Syllabic metre
- Quantitative metre
- Syllabic stress metre

The *syllabic stress metre* is the most common metre in English poetry. In this type of metre there are certain types of metrical feet defined by the syllable length and where the stressed syllable appears. Examples of the most used feet are:

- **Troschee**: 2 syllables, stressed — unstressed; e.g. incest
- **Iamb**: 2 syllables, unstressed — stressed; e.g. inject
- **Dactyl**: 3 syllables, stressed — unstressed - unstressed; e.g. terrible
- **Amphibrach**: 3 syllables, unstressed — stressed - unstressed; e.g. incumbent
- **Anapest**: 3 syllables, unstressed — unstressed - stressed; e.g. interrupt

2.2. Natural Language Processing

Natural Language Processing (NLP) is a field that covers computer understanding and manipulation of human language. It is a way for computers to analyze, understand and derive meaning from human language. Among a multitude of different areas within NLP, the following are particularly relevant for this project.

Tokenization

Tokenization is the task of finding words in a given document. More specifically, given a character sequence and a defined document unit, tokenization is the task of chopping it into pieces, called *tokens*. More often than not, one would at the same time like to throw away certain characters such as punctuation and exclamation marks. The *tokens* are often words or terms. The major question of the tokenization phase is what are the correct tokens to use. When glancing at the problem, it seems trivial to split at white space and chop away punctuation marks. However, this is not enough. In this Thesis' data set, a number of different words and forms of words occur like *sha-la-la* and *wikedely-spikedely*. Are these desirable words? And if so, do you split on the hyphen?

Part-of-Speech Tagging

When a text is tokenized, another important NLP task is to part-of-speech (POS) tag tokens. POS-tagging is the act of assigning part-of-speech tags (e.g. verbs and nouns) to each token of a sentence. Doing this is not a very simple task, because words have multiple meanings, words can be used as both nouns and verbs, in addition to other complications. For instance, look at this example:

- The last remark was an **insult**.
- How dare you **insult** me?

Here, the word **insult** is used both as a noun and a verb, thus when POS-tagging one need to take the whole sentence, and in some cases previous text, into consideration.

POS-tagging is a widely researched area, and the results from the field have been slowly improving over the past years. The results on the Wall Street Journal have been above 97% accuracy for tagging the correct word class for the last 7 – 8 years, and recently Choi (2016) received a new high of 97.64%. Huang et al. (2015) developed another high achieving algorithm reaching 97.55%.

2.3. Computational algorithms

There have been numerous different approaches to poetry generation. However, lately most research has been conducted within deep learning with focus on neural networks. Therefore, neural networks will also be the main algorithm for this Thesis. The same applies to object recognition in images, where convolutional neural networks are the best achieving algorithms for now.

2.3.1. Deep machine learning

Deep learning is a branch of machine learning which focuses on learning by extracting different features at different layers of the system and remembering them when similar features are seen again. More precisely, Deng et al. (2014) describe deep learning as a class of machine learning algorithms that share these traits:

- A system of multiple layers of nonlinear processing units to extract features and transformations. The layers in the system pass their output to the next layer as an input.
- The algorithms are based on the learning of multiple levels of features or representations of data. Higher level features are derived from lower levels.
- Learn multiple levels of representation that correspond to different levels of abstraction.

The first deep systems emerged in the 1960s: Ivakhnenko and Lapa (1966) published the first general, working learning algorithm for supervised deep feed forward multilayer perceptrons., while Ivakhnenko (1971) already described a deep neural network with 8 layers. Over the course of decades the field expanded and today it is one of the most popular fields within machine learning.

The convolutional neural network (CNN), which will be explained in detail later in this section, is a good example of a deep learning system. In short, it takes an input, e.g. an image of a zebra. Firstly, it iterates over parts of the picture to look at the patterns of the image. In this stage it finds high level abstractions such as the zebra's head and body. It then shrinks the window size of what it is looking at, identifying the zebra's eyes and mouth. Depending on the size of the CNN it can then look at even lower level abstractions. This is deep learning in a nutshell.

However, not all CNNs or other forms of neural networks are deep. If the CNN in the example above only looks at one level of abstraction from the zebra picture, it is called a *shallow* CNN. Many neural networks are shallow, if a network is shallow or deep mostly depends on how the network fulfills the points discussed in this section.

2.3.2. Artificial Neural Networks

Now that an introduction to deep learning has been given, it is time to look closer at artificial neural networks (ANNs). The first work related to artificial neural networks was McCulloch and Pitts (1943), which introduced an architecture similar to a shallow feed forward neural network. Subsequently, more work has been done to improve ANNs. Most notably, perceptrons got introduced by Rosenblatt (1958) and the popular backpropagation algorithm got introduced by Werbos (1975). These additions made neural networks a lot more viable, because of the computational power the backpropagation algorithm saves.

ANNs are machine learning models that are often used as prediction models or classifiers. The idea behind ANNs is that they try to model the way the human brain solves problems. They consist of neurons in layers connected to layers behind and in front of them. The input comes in through the first layer, called the input layer. The data travels from layer to layer and is processed by the neurons in each layer before being passed on to the next. The final layer is called the output layer, which gives a prediction depending on the initial problem. ANNs strengths are that they are able to learn very complex features, and in some fields, such as image recognition, they perform very well. However, they are hard to control because of the huge internal

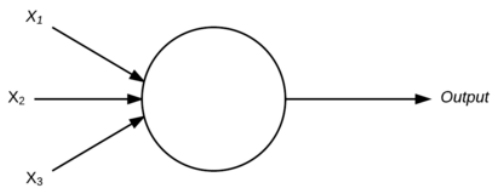


Figure 2.1.: The most basic form of the perceptron

structure they have, and sometimes it might be difficult to see what really happens.

Before going more in depth of ANNs, we need to delve into more depth of the core of neural networks — the perceptron.

The perceptron and artificial neurons

The perceptron is the most basic unit of a neural network. A perceptron in its simplest form takes one or several binary inputs x_1, x_2,..., and produces a single binary output. Figure 2.1 shows a basic perceptron. In addition, Rosenblatt proposed a simple rule to compute the output. He introduced *weights,* w_1, w_2,...,. These weights are multiplicated to their respective input, and if the sum is higher than some threshold, the perceptron returns 1, else 0. This is called the *activation function* of the perceptron. Equation 2.1 is the simple perceptron activation function.

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq threshold \\ 1 & \text{if } \sum_j w_j x_j > threshold \end{cases} \tag{2.1}$$

Today, this form of the perceptron is rarely used. Instead, a more sophisticated unit called *artificial neuron* is preferred. The input of the neuron is not a binary value of 0 or 1, it is a continuous value between 0 and 1. Multiple different types of neurons are available, depending on the type of activation function used in the neuron. Table 2.1 shows a few different activation functions to give the reader a feeling of how the mathematics is calculated. This Thesis is however not going into more detail as it is out of its scope. Convolutional and recurrent neural networks often use a different setup, which will be explained in detail in their respective sections.

Table 2.1.: Three different activation functions

Name	Equation
Sigmoid	$\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}}$
Rectifier linear unit (ReLU)	$f(x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases}$
tanh	$f(x) = tanh(x) = \frac{2}{1+\mathrm{e}^{-2x}} - 1$

Feed forward neural networks

Feed forward neural networks are the most basic form of neural networks. These networks consist of one input layer, and one output layer. Between the input and output layer, there are typically one or more hidden layers. Figure 2.2 shows a basic feed forward neural network with one hidden layer. The input layer represents the input data, for instance a pre-processed image or vector representing words. The data from the input layer travels to the next layer, more often than not a hidden layer. As we know, this data gets processed by the activation function of the neurons in the hidden layer. When the activation functions are done calculating their output, the information travels to the next layer. If the next layer is an output layer, such as in Figure 2.2, the data gets processed by the last activation function of the network.

The activation function of the output layer has the same basic principle as regular activation functions, where multiple inputs x_1, x_2,..., are taken as input from a previous layer and calculates an output. However, the last function is able to give a final prediction or classification for the network. A very common output function is the softmax classifier, described in Equation 2.2.

$$\sigma(x)_\mathrm{j} = \frac{\mathrm{e}^{x_\mathrm{j}}}{\sum_{k=1}^{K} \mathrm{e}^{x_\mathrm{k}}} forj = 1, ..., K \tag{2.2}$$

where K is the dimension of the output vector. The functions shown in Table 2.1, such as Sigmoid and tanh can also be used as output functions.

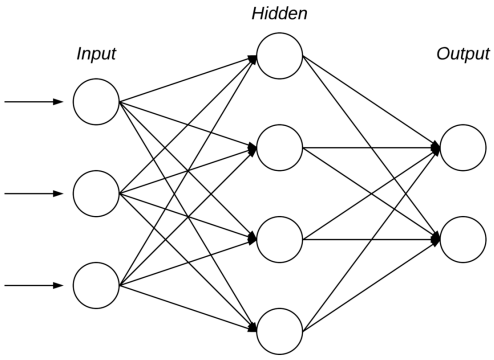


Figure 2.2.: A Feed Forward Neural Network

Now, recall Equation 2.1. Since the notion of $\sum w_j x_j \leq threshold$ is quite cumbersome, two changes are made. One is to write $\sum w_j x_j$ as the dot product, $w*x = \sum w_j x_j$ where w and x are vectors whose components are the weights and inputs, respectively. Another change is to move the threshold to the other side of the equation, and replace it by what is known as the perceptron's bias, such that $b = -threshold$. The new equation is now

$$output = \begin{cases} 0 & \text{if } w * x + b \leq 0 \\ 1 & \text{if } w * x + b > 0 \end{cases} \tag{2.3}$$

Another factor worth noticing is the connectivity between layers. If all neurons in layer x are connected to all neurons in layer *x+1*, layer x is said to be fully connected to layer *x+1*. Partially connected networks are also common, it usually comes down to the goal of a network and its architecture.

Loss functions are indications of how well a model is performing. Often the model makes improvements and learns because it tries to minimize the value of this function. Different loss functions are used in different settings. If you work with classification of images, a normal loss function is a variation of the softmax classifier which makes use of cross entropy loss. The idea of cross entropy is simply just to measure how many correct and false predictions the model made to give a hint of how the model is performing. In the case of regression, where the model is to predict a continuous value, a common loss function is simply to get the model's prediction distance to the real value and then regularize it using regularization techniques. By regularizing the loss function, it makes the network prefer smaller weights rather than large weights. The result of this regularization is that it helps prevent overfitting on the training data.

As stated in the previous section, the goal of a feed forward neural network is to minimize the loss function, by updating its weights and bias. This is often done using backpropagation with an optimization method such as gradient descent. The algorithm repeats a two phase cycle. First it computes the propagation and then it updates the network's weights. Secondly, an input is presented to the network, propagated forward layer by layer until it reaches the output layer. The output error is calculated by a loss function described in the previous paragraph. The error value is then propagated backwards, until each neuron has an associated error value which roughly represents its contribution to the original output. The backpropagation algorithm then uses these error values to calculate the gradient (derivative) of the loss function. Finally, this gradient is fed to the optimization method and updates the weights in the network. Now, the network is ready for another input, and calculates the loss of this new input.

A final feature to notice is the dimension of the input and output layers. The dimension of the input layer often heavily depends on what form the data is. For example, if you have images that are *100 x 100* pixels in size, and each pixel is represented by a vector of *r-b-g*, then the total size of one image is *100 x 100 x 3* which is *30,000*. This would be the size of the input layer. The output layer depends on the function of the network. If the network is classifying the images into 10 classes, then the output size should be *10*.

Now that the artificial neurons and a shallow feed forward neural network have been explained, we can proceed to the two neural network architectures used in this Thesis: the convolutional neural network and the recurrent neural network.

Convolutional neural networks

One of the pioneer works in the world of Convolutional Neural Networks (CNNs) was LeCun et al. (1998), creating LeNet-5 that classifies handwritten digits, used today in several banks. CNNs are often used in image recognition because of their ability to handle two dimensional input.

CNNs are a bit different than feed forward neural networks. The main difference is the introduction of two new types of layers. In addition to fully connected layers, the main idea behind a CNN is represented in convolutional and pooling layers. If you think about it, a feed forward neural network does not take the spatial structure of an image into consideration. It treats pixels that are close together and far apart the same. On the other hand, a convolutional layer takes this into consideration. Three keywords are important in CNNs: *local receptive fields*, *shared weights* and *pooling*.

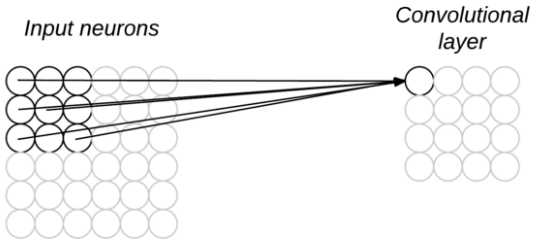


Figure 2.3.: This figure shows the mapping from the input layer to a convolutional layer. The window size here is *3 x 3*. For the next neuron the window is shifted one step to the right, so that the neuron in the convolutional layer looks at a different part of the image.

Local receptive fields mean that instead of making a fully connected layer, we only make connections in small, localized regions of the input image. Given a *5 x 5* region in the convolutional layer, the layer now looks at 25 input pixels. However, all of these are close together in the image. Figure 2.3 showcases this. The region highlighted is the local receptive field. The window size varies, and is often larger the earlier the layer is in the architecture. The window then slides across the entire picture, covering the whole input picture. One can think of it as certain neurons in the hidden layer only learn about certain parts of the picture. This makes the convolutional layer able to extract features from the picture.

As previously stated, each neuron in the first convolutional layer looks at a window in the input image. If we recall Equation 2.3, each neuron has its weights and biases, calculating the input from the previous layer. In convolutional layers, however, the weights and bias are now shared. When the window slides to the next section of the input layer, the weights stay the same, while only the values of the pixels change. In theory, this means that all neurons in the convolutional layer detect exactly the same feature. For this reason, we call the mapping from the input to the convolutional layer a *feature map*. It is not enough to only have one feature map, multiple feature maps are commonly used. A big advantage with shared weights is that they greatly reduce the number of parameters involved. In a *5 x 5* feature map, only *25 + 1* parameters are involved.

In addition to convolutional layers, CNNs also contain *pooling layers*. These layers almost always succeed the convolutional layers. The use of pooling layers is to simplify the information of convolutional layers. The pooling layer takes each feature map output from the convolutional layer

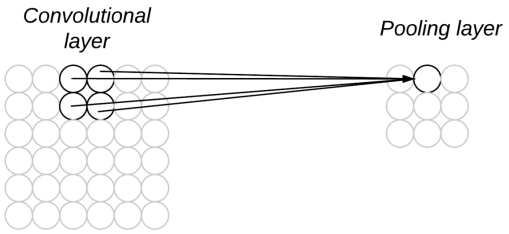


Figure 2.4.: An example of a pooling layer. This is a textbook *2 x 2* size max-pooling.

and creates a condensed feature map. A normal size region to pool from is a *2 x 2* window. Now, from here it depends on what kind of pooling one uses. In *max-pooling* you take the four neurons and return the highest value and save it. Thus, the size of the new layer is a lot smaller and a compressed version of the convolutional layer. Figure 2.4 shows an example of pooling.

This sums up convolutional neural networks. It is normal to have multiple fully connected layers, convolutional layers and pooling layers, all depending on the purpose of the network.

Recurrent neural networks

The last type of neural networks this Thesis will cover is the recurrent neural network (RNN). One of the first documented usages of RNN architecture is Hochreiter (1991). Today, poetry generators often use deep recurrent neural networks because of their ability to handle sequences.

You can think of sentences as sequences of words. RNNs make use of such sequential information. Traditional feed forward neural networks assume all inputs are independent of each other. However, for many tasks, such as language modeling, that is a bad idea. If you want to predict the next word in a sentence, it would help to know the entire sentence, not only the previous word. RNNs are called recurrent because they perform the same task for every element in a sequence, with the output being dependent on all previous computations. Figure 2.5 shows the main idea behind a recurrent neural network, where the input of the calculation for the next hidden state is the previous hidden state and the current input. The box labeled A is what calculates the hidden state.

The hidden state in a RNN is computed as follows:

$$h_{\mathrm{t}} = \phi(W\mathbf{x}_{\mathrm{t}} + U\mathbf{h}_{\mathrm{t\text{-}1}}) \tag{2.4}$$

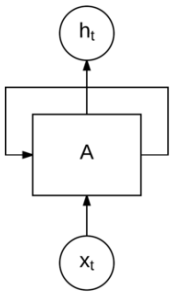


Figure 2.5.: The idea behind a recurrent neural network. The hidden state at time t is dependent on both the input at time t and the hidden state at time *t-1*.

The hidden state, h, at time t is a function of the input at the same time step $\mathbf{x}_{\mathrm{t}}$, modified by a weight matrix W added to the hidden state of the previous time step $\mathbf{h}_{\mathrm{t\text{-}1}}$ multiplied by its own hidden-state-to-hidden-state matrix U. ϕ is the squashing function — either a logistic sigmoid function or tanh, described in Table 2.1.

Another type of RNNs is long short-term memory (LSTM) networks that were introduced by Hochreiter and Schmidhuber (1997). They are able to work with much longer sequences than regular RNNs, and thus is what this Thesis has chosen to work with as a predictor on the poetry generation part. Regular RNNs are able to only work with a few steps back, but LSTMs are able to look thousands of steps behind. Figure 2.6 shows the architecture of an LSTM. From the figure, you can see how the input x_t is processed using different equations, and the hidden state h_t is returned and passed into the next step of calculating x_{t+1}. These equations are explained next.

LSTMs also have the chain-like structure of a RNN, but the calculations for the hidden state are a bit different. In total, five equations are used to calculate the new hidden state. In the following text, the same notation as describing a regular RNN is used, and $\circ$ represents the Hadamard product. Furthermore σ represents activation functions, where σ_{g} is the sigmoid function and σ_{c} is the hyperbolic tangent.

The first part is to calculate the forget gate vector, f_t. It corresponds to the weight of remembering old information. It is very similar to how updating works in a regular RNN, in addition to adding a bias for the forgetting gate, b_f. The equation is:

$$f_{\mathrm{t}} = \sigma_{\mathrm{g}}(W_{\mathrm{f}}\mathbf{x}_{\mathrm{t}} + U_{\mathrm{f}}\mathbf{h}_{\mathrm{t\text{-}1}} + b_{\mathrm{f}}) \tag{2.5}$$

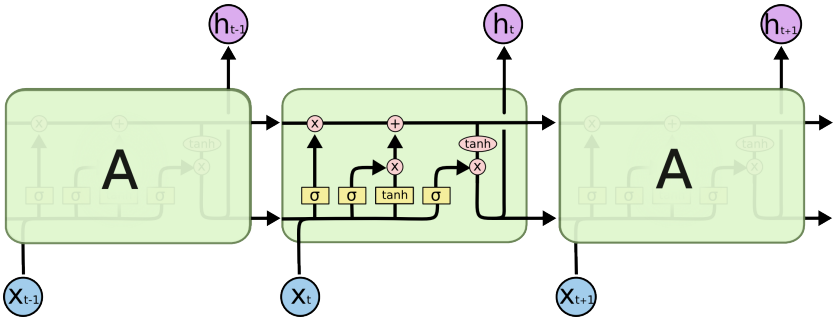


Figure 2.6.: An LSTM chain. The rectangle in the middle illustrates an architecture showing how an LSTM unit does its calculations. Taken from http://colah.github.io/posts/2015-08-Understanding-LSTMs/ with permission from Cristopher Olah.

The next part is to calculate the input gate vector, i_t. This is the weight of acquiring new information. The calculations are the same as above, one just has to exchange the matrices to the corresponding gate. The formula is:

$$i_{\mathrm{t}} = \sigma_{\mathrm{g}}(W_{\mathrm{i}}\mathbf{x}_{\mathrm{t}} + U_{\mathrm{i}}\mathbf{h}_{\mathrm{t\text{-}1}} + b_{\mathrm{i}}) \tag{2.6}$$

Thirdly, the output gate vector o_t is calculated. Again, one only has to change to the corresponding matrices:

$$o_{\mathrm{t}} = \sigma_{\mathrm{g}}(W_{\mathrm{o}}\mathbf{x}_{\mathrm{t}} + U_{\mathrm{o}}\mathbf{h}_{\mathrm{t\text{-}1}} + b_{\mathrm{o}}) \tag{2.7}$$

Second to last, the cell state vector needs to be updated. This is done by taking the Hadamard product of the forget gate vector and the previous cell state vector and adding those to the Hadamard product of the input gate vector and Equation 2.7 with matrices corresponding to the cell state vector.

$$c_{\mathrm{t}} = f_{\mathrm{t}} \circ c_{\mathrm{t\text{-}1}} + i_{\mathrm{t}} \circ \sigma_{\mathrm{c}}(W_{\mathrm{c}}\mathbf{x}_{\mathrm{t}} + U_{\mathrm{c}}\mathbf{h}_{\mathrm{t\text{-}1}} + b_{\mathrm{c}}) \tag{2.8}$$

Finally, the new state can be calculated. This is done by taking the Hadamard product of the output gate vector at time t and an activation function, usually $\sigma_{\mathrm{h}} = x$, multiplied by the new cell state calculated above.

$$h_{\mathrm{t}} = o_{\mathrm{t}} \circ \sigma_{\mathrm{h}}(c_{\mathrm{t}}) \tag{2.9}$$

Now that neural networks and how they work have been described, the base for this Thesis' algorithms is laid. These algorithms do most of the heavy lifting, both for the poetry generation and the image recognition part of this Thesis.

2.4. Evaluation metrics of poetry

This section discusses four different methods of evaluating poetry used in state-of-art poetry generation today. The two first ones, BLEU and METEOR are machine based approaches while the two last ones are human based evaluation.

Bilingual evaluation understudy (BLEU) is a method for automatic evaluation of machine translation (Papineni et al., 2002). The main idea behind the BLEU measure is to take two different system translations A and B from a source sentence S, together with multiple reference lines written by humans, say reference translations C, D and E. Now lines A and B, which are translations that one wants to compare and find the better one, will be compared to lines C, D and E. Whichever of the translations A and B's n-grams were closer to the n-grams of the human written lines C, D, E, are considered better translations. BLEU does not take the position of the words into consideration, thus they are unordered. A problematic aspect of this evaluation is that human written references are needed, and those are not available for this Thesis.

A second machine based evaluation method is the METEOR evaluation (Banerjee and Lavie, 2005). It was designed to explicitly address several observed weaknesses in the BLEU metric. In addition to what the BLUE metric measures, the METEOR evaluation is able to match words that are simple morphological variants of each other (i.e. they have identical stem). METEOR is also able to find synonyms, not only identical words. The core idea of METEOR is that, given a system translation and a reference translation, it creates an alignment between the two strings. An *alignment* here is a mapping between unigrams such that unigrams in one translation can map to zero or one in another reference, and zero times to its own. These alignments can then be used as a measurement of how good a translation is. However, the same problem as before is present, one needs human generated poetry for exactly the same pictures this Thesis' system computes from.

Another problem with these evaluation methods are that later research (Liu et al., 2016; Wang et al., 2016b) has found that these methods have little coherence with human evaluation. Therefore, this Thesis' focus will lay on human evaluations described next.

A commonly used method to evaluate poetry is by the criteria *grammaticality*, *meaningfulness* and *poeticness* (Manurung, 2004). Manurung defines each of them as the following:

- **Grammaticality**: A poem must obey linguistic conventions that are prescribed by a given grammar and lexicon. Basically, the text gener-

ated must lexically and syntactically make sense for the human brain. However, in poetry, sometimes the poet intentionally make grammar mistakes. Thus, the goal of Manurung is to essentially rule out random sequences of words.

- **Meaningfulness**: This property states that the poem must convey some conceptual message which is meaningful under some interpretation. In this Thesis it will also be a measurement of how the system can keep on topic with the visual input.

- **Poeticness**: Here, poeticness refers to phonetic features such as rhymes and rhythmic patterns.

The scales on which these three dimensions are measured, in Manurung (2004), are the following:

- **Grammaticality**
  3. Grammatically correct
  2. Partially grammatically correct
  1. Not correct

- **Meaningfulness**
  3. Meaningful
  2. Partially meaningful
  1. Not meaningful

- **Poeticness**
  3. Poetic
  2. Partially poetic
  1. Not poetic

Each of the dimensions is scored on a scale of 1 – 3, 3 being the highest score, and 1 being the lowest score. A poem is a text fulfilling all of these properties of meaningfulness, grammaticality and poeticness.

However, lately a slightly different version of these criteria has been introduced (Yan et al., 2013; He et al., 2012; Zhang and Lapata, 2014; Wang et al., 2016b). These criteria are *fluency*, *meaning*, *poeticness* and *coherence*. These measurements ask the following questions:

- **Fluency**: Does the poem follow the rhyme and tone requirements?

- **Meaning**: Does the poem read smoothly and fluently?
- **Poeticness**: Is the poem coherent across lines?
- **Coherence**: Does the poem have a certain meaning and artistic conception?

Although these criteria also could work nicely, an in-depth explanation of these criteria have not been made. Furthermore, the different sources listed above seem to use slightly different definitions of the four metrics. Hence, this Thesis will use the metrics of Manurung (2004) with the scales described earlier.

2.5. Frameworks

This section introduces different frameworks used in this project. A variety of APIs are used, thus giving a short introduction of them is necessary. Only parts of the frameworks that are relevant to this Thesis will be introduced, therefore big parts of the frameworks might be ignored.

2.5.1. ConceptNet

ConceptNet[1] (Speer and Havasi, 2012) is a freely available semantic network, created to help computers understand the meaning of the words people use. It started as a crowd-sourcing project named Open Mind Common Sense, which was launched in 1999 at the MIT Media Lab. It has since grown by integrating knowledge from other crowd-sourced resources.

In ConceptNet each word or phrase is divided into nodes called *concepts*. The links between these nodes are called *edges*. For instance, if you look up *zebra*, you will find it is a node. A node zebra is linked with is the node *zoo*. The link, or edge, between them is in this case called *AtLocation*. ConceptNet is made up of thousands of these nodes and edges, helping computers and researchers use semantics in their work.

Multiple different API calls are available with ConceptNet, but this Thesis uses two of them: Asking for all information of a given concept, and asking for only relevant concepts given a concept. The two calls return mostly a similar result, but often one of the calls will give some outputs that the other call did not consider, which makes both calls useful to gather as many possible links from different concepts as possible.

[1] `https://www.conceptnet.io/`

2.5.2. TensorFlow

TensorFlow[2] is an open source framework for machine intelligence in Python, made by Google. It was developed by researchers and engineers working on the Google Brain Team for conducting machine learning and deep neural networks research. With the TensorFlow API one is able to use Google's convolutional neural network, Inception.

The main properties of TensorFlow are Tensors. These tensors are where all information is stored. For instance, one tensor is created for the input, and another one for the output, and others wherever different information needs to be stored. A TensorFlow program consists two phases. In the first phase you build the model, i.e. you set up each layer and which optimizer to use. In the second phase you run the graph with the desired functions and get the calculations and output. TensorFlow offers many different kinds of services, but the most important one for this Thesis is the way TensorFlow creates RNNs and CNNs. The architecture for the RNN used in this Thesis is discussed in Chapter 5.

An interesting addition to TensorFlow is the NVIDIA CUDA Deep Neural Network[3] (cuDNN) library helping computation on GPUs. cuDNN is available for multiple frameworks like TensorFlow, Caffe, Torch and Theano. It is not exclusive to TensorFlow. Since computations in neural networks are very heavily matrix based and GPUs are built to handle simultaneous calculations, the computational power increases a lot with the cuDNN library installed.

Inception

Szegedy et al. (2016) made Inception-v3 which is the latest model of Google's convolutional neural network for image recognition. The model is pre-trained and available through TensorFlow. The source code is also available, so tweaking it for other image recognition purposes is possible.

Beneath is a table showing the power of Inception-v3. It was tested using four models, and 144 different crops were evaluated. It achieved very high performance on both Top-1 and Top-5 accuracy. Top-1 accuracy is the percentage where the correct classification is the class output by a system. Top-5 accuracy is the percentage where the correct classification is in the top 5 most likely classifications made by a system.

In this Thesis, Inception is the first step of the system. This is for multiple reasons: Firstly, because the performance of Inception is one of the best in

[2] `https://www.tensorflow.org/`
[3] `https://developer.nvidia.com/cudnn`

Table 2.2.: Inception-v3 performance. The data is from Szegedy et al. (2016)

Network	Models Evaluated	Crops Evaluated	Top-1 Error	Top-5 Error
Inception-v3	4	144	17.2%	3.58%

the world. Secondly, because it is available through TensorFlow which is used for this project, and thus fits nicely in. Furthermore, training a new CNN is time consuming and will (probably) not yield as good results as Inception because of the computational power available. Lastly, it is possible to retrain Inception so you can add more classes if it is necessary or desirable for the system. Thus, Inception seems like a good choice for the image recognition part of the system. However, there is one downside: If the input picture does not contain any elements of the 1,000 classes it is trained on, the system is going to perform poorly. A more in-depth explanation of Inception's architecture is available in Chapter 5.

2.5.3. Natural Language ToolKit

The Natural Language ToolKit (NLTK)[4] (Bird et al., 2009) is a platform to work with human language data. It provides interfaces to over 50 corpora, and other lexical structures such as WordNet. It also provides text processing libraries for classification, tokenizing, stemming, and parsing text. Some of the features from the NLTK platform are used for this Thesis and described below.

The first couple of features are the tokenization process and tagging of words. Both processes are described in Section 2.2. NLTK also includes support for regular expressions, which can get rid of unwanted characters or sequences of characters. POS-tagging of the words also comes in handy when you want to analyze a given sentence.

Another feature used is the Carnegie Mellon University Pronouncing Dictionary [5] (CMUdict). CMUdict is an open-source machine-readable pronunciation dictionary for North American English that contains over 120,000 words. CMUdict works as follows: The word *lexical* is in the CMUdict represented as *L EH1 K S IH0 K AH0 L*. This is useful in this project for two reasons: Rhythm and Rhyme. The integers in *lexical* represent the stress for all syllables, thus the rhythmical pattern for *lexical* is *100*. Here, the 0 represents no stress, the 1 represents primary stress, and 2 represents secondary stress. In this Thesis though, 2 will be merged with 1 to just represent stress for simplicity. The second use is for rhymes, where the last part of words

[4] `http://www.nltk.org/`

can be compared to calculate rhyme words. Based on the rules of rhyming in Section 2.1, it is possible to check that the ending of the word is the same (not including the stress), and that the preceding consonant is different.

[5] http://http://www.speech.cs.cmu.edu/cgi-bin/cmudict/

3. Related Work

In this chapter, the state-of-the-art poetry generation and image classification are presented. Poetry generation started developing as a research field in the 1990s, with multiple different methods being tried out, the latest using deep learning. Deep learning has also become very helpful in image recognition where convolutional neural networks have heavily dominated the latest years. The main focus will be on poetry generation because that is what the majority of this Thesis' system consists of.

3.1. State-of-the-art poetry generation

The high complexity of creative language creates substantial challenges for poetry generation. But even though the task is complex, many interesting poetry generation systems have been developed. Gervás (2002) roughly divides poetry generation into four different groups:

- Template Based Poetry Generation.
- Generate and Test Approaches.
- Evolutionary Approaches.
- Case-Based Reasoning Approaches.

In 2002 these might have been accurate, but today they are a bit outdated. In addition to these four groups, *Corpus-based methods* and *Deep Learning methods* are the two others discussed in this Thesis, as they are the dominant poetry generation techniques used today.

3.1.1. Template Based Poetry Generation

Template based poetry generation makes use of incomplete templates and fills them with words from a dictionary to suit a set of defined syntactic and/or rhythmic constraints. They are simple to build, but the creativity of the computer is debatable: the computer only follows a set of rules.

One system using this approach is the Poetry Creator. According to Oliveira (2009), Poetry Creator fills a predefined poem template with words that describe a certain subject, a synonym for that subject, and a title for the poem. All of these are provided by the user.

PoeTryMe (Oliveira, 2012) is another system using this approach. The system takes advantage of a sentence generator to obtain lines and to build a poem. The poem is generated according to a set of seed words, used to get sentences from the sentence generator, and a poem template. The template contains information such as the poem's structure, number of stanzas, number of lines per stanza, and number of syllables per line.

The sentence generator is the core of PoeTryMe, and is made primarily with a semantic graph which finds related words, and a grammar generator which contains templates for the grammar in the poems. So with the sentence generator and poem templates, poetry is created.

A third system based on this approach is Netzer et al. (2009). This system uses Word Association Norms (WANs) as a lexical knowledge source and investigates the Haiku genre. The system involves five steps: theme selection, syntactic planning, content selection, filtered over-generation, and ranking.

The theme selection takes in a user-supplied seed word and finds various associations. Different heuristics are used, with the most successful heuristic using an associations graph and conducting a short random walk on this graph. The next step is the syntactic planning which determines the form of the Haiku by setting various constraints on the system. The third step, content selection, looks for Haiku lines associated with the theme and fitting the syntactic structure. Generation is the fourth step and involves creating many possible Haiku candidates by randomly collecting lines created in the previous step. Finally, the last step ranks these Haikus, where the top ones are chosen as the output of the system.

3.1.2. Generate and Test Approaches

Generate and test approaches are also perceived to be fairly simple: random word sequences are produced according to formal requirements such as metrics and semantic constraints. The produced word sequences are either accepted or rejected. If rejected, a new word sequence is generated.

One of the early attempts at generate and test approaches was Manurung (1999)'s chart system. The chart generation generates syntactically correct natural language strings that satisfy rhythmic patterns. Usually a parser analyses strings and translates them into logical forms. However, Manurung's chart generator translates logical forms into strings. The input semantics are described by first order predicates that logically represent sentences. When

new rules are obtained, fitting stress patterns are created with the help of pronunciation dictionaries. Randomly created words are then being checked if they fit into the stress patterns, and added if they do. The system manages to maintain the syntactic, semantic and rhythmic well-formedness, at the cost of being very computationally expensive and not very flexible because it generates a perfect result or no result. This system was used within Manurung (2004).

Another early attempt was Gervás (2000) called Wishful Automatic Spanish Poet (WASP). WASP is a forward reasoning rule-based system that was considered one of the first serious attempts on automatic poetry generation. It aims to study and test the importance of the initial vocabulary, word choice, pattern selection and construction heuristics for accepting poems created by this system. The basic verse generation algorithm starts with the selection of the appropriate pattern. Then, a word corresponding to the first category of the pattern is randomly chosen and appended to the draft. At each of these stages, the poem is being tested against the requirements. If the conditions are not met, the verse is rejected and a new verse starts being generated. The produced poetry conformed to the formal metrics like rhythm and rhyming, but it made little sense from a linguistic point of view.

Oliveira et al. (2007) introduced Tra-La-Lyrics. Tra-La-Lyrics aims to generate text based on the rhythm of a song's melody. By using the beat of a song as rhythmic patterns, poetry generation is almost identical to lyric generation. Three strategies were developed in order to generate lyrics: Random words, generative grammar, and the generate and test aspect. When choosing the words in the lyrics, the only constraint is rhythm, thus any word satisfying the rhythm can be chosen. Generative grammar ensures that the word choice does not only satisfy rhythmical constraints, but also syntactical. The latter takes priority over the former. If neither can be satisfied, backtracking is used. The last strategy is the generate and test strategy. Words following sentence templates are generated and evaluated against musical sentences. After multiple generations, the most fitting sentence is chosen.

3.1.3. Evolutionary Algorithms

Evolutionary computing follows techniques from the biological evolution, such as natural selection and genetic inheritance. These properties can be used for computational poetry generation, and there have been a couple of attempts of using evolutionary algorithms. However, before describing these, a brief introduction to evolutionary algorithms is presented.

Michalewicz (1994) defines an evolutionary algorithm as a multi-point stochastic search algorithm. This means that the search algorithm uses a

heuristic that explores multiple points in the search space simultaneously and can avoid getting trapped in a local maximum. A basic evolutionary algorithm consists of five steps:

1. **Initialization**: Create a new population representing a set of starting points randomly spread out over the search space.
2. **Evaluation**: Evaluate each possible solution, meaning measuring its fitness.
3. **Selection**: Form a new population by stochastically selecting individuals based on the fitness score of the old population.
4. **Evolution**: Transform some of the members of the new population through some ground rules, resulting in new individuals being born.
5. **Repeat**: Step 2 to 4 until either:
   a) x number of iterations has occurred, or
   b) x fitness score is reached, or
   c) the algorithm converged

Levy (2001) created POEVOLVE, a computational model of poetry generation, creating limericks based on evolutionary algorithms. POEVOLVE takes the real process of human poetry writing as a reference to draw the intuitions of the system. It uses the following architecture consisting of four modules:

- One or several generator modules: This is to create the initial population of possible poetic objects and modify them in later generations.
- Evaluator modules: To analyze and select the best fitness score.
- A work space where the population resides.
- Two knowledge bases: One conceptual and one syntactical.

The generator modules create the initial population of candidate poetic objects and modify them in later generations. The evaluator modules analyze the individuals at each generation, and rank them based on fitness. The work space is where the population resides. The knowledge base includes a lexicon, a conceptual knowledge base, and a syntactical knowledge base.

The initial population is created from a set of words that include phonetic and stress information. Appropriate rhyming words based on rhyme patterns

are selected and a genetic algorithm is employed. Mutation is achieved by crossover operators that modify the words in the population, and the fittest population are chosen for the next generation. The evaluation is performed by a neural network trained on human judgments on how creative each limerick is. This prototype, however, does not take syntax and semantics into consideration.

A second model is called McGonagall (Manurung, 2004). As mentioned earlier, this model is based on Manurung's own chart system. Manurung formulated poetry generation as a state space search using stochastic hill-climbing search, where a state in the space is possible text and a move can occur at any level, i.e. from semantics to phonetics. The system generates metrically constrained poems based on a given topic using a grammar-driven formulation.

The stochastic hill-climbing search is an evolutionary algorithm with two stages: Evaluation and evolution. An initial population based on phonetics and semantics is created. Every individual in the population are then scored and the fittest individuals are drawn out to mutate into a new generation. Each individual is scored at each step and the highest scoring ones go to the next generation, which is the evolution process. Since the mutation can happen at any level of representation, operators must preserve consistency.

The system reaches a goal when all of the constraints of meaningfulness, grammaticality and poeticness described in Section 2.4 are fulfilled. According to Manurung et al. (2012), the McGonagall system is capable of finding optimal solutions in decent-sized search space for semantics and metre pattern separately, but when these properties are looked at simultaneously, the system is having some difficulties.

3.1.4. Case-Based Reasoning (CBR) Approaches

Case-based reasoning is another popular approach to poetry generation. CBR approaches in poetry generation first retrieve existing poems and adapt them based on a target message provided by the user to fit the content.

Gervás (2001) proposed the ASPERA system, which is a forward reasoning rule-based system. The system drafts a poem and asks for validation from the user when given a short description of the poem. ASPERA (Automatic Spanish Poetry Expert and. Rewriting Application) is an evolution of the WASP system Gervás (2000) described earlier, improving the system implemented in WASP.

The generation process starts by interacting with the user to obtain features for the poem, for example rhyme structure, mood (positive or negative), and the length of the poem. These configurations are then used to search in

the knowledge base for the most appropriate strophic form and vocabulary. The user also provides a prose paraphrase of the intended message that will be used in the planning of the poem draft.

Each line is generated with a four-step CBR approach:

- The retrieve step takes each sentence in the intended message and retrieves a specific verse from a corpus
- The reuse step takes the verse template and constructs a line based on the part-of-speech structure of the verse
- In the revise step, the user is asked to validate the draft
- And lastly, in the retain step, validated poems are analyzed and stored, making it possible to use them later

This system lacks strong poeticness because it selects words based only on their syntactic category and ranking with respect to the user-defined meaning, and not the metric information in the word choice. Gervás suggests improving the system by addressing this issue in addition to look at semantic information through a knowledge rich ontology.

Another system using a CBR approach is COLIBRI (Díaz-Agudo et al., 2002). This system is very similar to the ASPERA system, but the various cases are stored in very flexible representations, using a Description Logic System. COLIBRI also incorporates CBROnto, an application-dependent ontology, that improves the inference power of the system and the representation, and uses of more explicit and general knowledge.

The algorithm requires a list of keywords representing meaning and a specification of a strophic form as input, and then retrieves a case following the appropriate strophic form from a corpus of existing poems. Next, the user-defined keywords are replaced in the text, while conserving the syntactic well-formedness. Lastly, words are being swapped out to ensure the constraints regarding metre and rhythm are satisfied. A negative side to this is that the last step might destroy the intended meaning in order to satisfy the goal of strophic form.

3.1.5. Corpus-based Methods

Corpus-based methods are based on finding other poems, and use their structure to create new poems. They often use multiple corpora and often substitute words based on their POS-tag and relevance. This approach has developed more in recent years.

Colton et al. (2012) present Full-FACE Poetry Generation. The system is a corpus-based poetry generation system which uses templates to construct poems according to constraints on rhyme, metre, stress, sentiment, word frequency and word similarity. The system creates a mood of the day by analyzing newspapers. The algorithm consists of four steps:

- **Retrieval**: The system mines similes from the internet depending on the sentiment and evidence
- **Multiplication**: Objects, words and phrases are then substituted to create variations of the similes
- **Combination**: Similes and their variations extracted are then plugged into a template given by the user
- **Instantiation**: Random phrases are then chosen from and elaborate set to fill the fields of a user-given template

An aesthetic is created based on the article, and the system searches for an instantiation of the template maximizing the aesthetic. While doing this it provides commentary to add value to the creative act. Colton et al. (2012) argue that this is the first poetry system generating all aspects from scratch.

A system using two corpora is Toivanen et al. (2012). The corpora used is a grammar corpus and a poetry corpus. The reason for this is two-folded. Firstly, to provide semantic content for new poems, and secondly to generate a specific grammatical and poetic structure.

The system starts by choosing a topic, specified by a single word. Topic associated words are then extracted from a background graph. A background graph is a network of associations between words, based on term co-occurrence. The next step in Toivanen et al. (2012)'s system is selecting a desired length text randomly from the grammar corpus. This text is then analyzed and POS-tagged. Each word is then independently substituted, by words associated to the topic. At the end, after all words have been processed, the novelty of the poem is measured by looking at the percentage of replaced words. If more than half the words are replaced, the poem is accepted. Toivonen et al. (2013) further build on this model and show how simple methods can build surprisingly good poetry.

3.1.6. Deep Learning Methods

In recent years deep learning has become quite popular for poetry generation, and multiple studies have been done combining deep learning and poetry

generation. Many of these systems have shown promising results, which is the main reason why this Thesis also have chosen deep learning as the creative driving force.

Zhang and Lapata (2014) is one of the earliest attempts at generating poetry using deep learning. As most previous research, the poem is made by interacting with the users, e.g. the user provides different keywords. The generator then expands these keywords into a set of related phrases. The keywords, however, can only be words that appear in the *ShiXueHanYing* poetic phrase taxonomy, or at least that is what the system assumes. The generator creates the first line of the poem using these keywords. It selects all phrases in the taxonomy related to the keywords and ranks them using multiple character-level neural networks.

The first step is to use a convolutional sentence model (CSM) to convert lines into vectors that can be fed further into the system. The next step is a recurrent context model (RCM) that calculates the context of each of the words. The output of the RCM is then used as input in a recurrent generation model (RGM) to find the next character. This system is very complicated using a CNN and two RNNs, and is very computationally heavy, but yields respectable results. In 2014 this was one of the highest achieving system to date based on BLEU scores, according to Zhang and Lapata (2014).

Wang et al. (2016a) also propose an architecture using neural networks. They use an attention-based recurrent neural network, which accepts a set of keywords as the theme and generates poems by looking at each keyword during the generation. They are also using other techniques to improve the model, including character vector initialization, attention to input, and hybrid-style training.

The core of the system is the attention-based sequence-to-sequence model (Bahdanau et al., 2014). Firstly, the input sequence is converted by an encoder to a sequence of hidden states to represent the semantic status at the position of the input. These hidden states are then used to regulate a decoder that generates the target sequence. The encoder is a bi-directional GRU that converts the input keywords into a sequence of hidden states. The decoder then generates the whole poem character by character. At each time step t, the prediction for the next character y_t is based on the current status of the decoder as well as all the hidden states of the encoder.

A big upgrade from the system of Zhang and Lapata (2014) is that in Wang et al. (2016a)'s system a GRU model is preferred in both the encoder and decoder. Zhang and Lapata (2014) use a vanilla RNN. A vanilla RNN, as described in Section 2.3, often forgets sequences multiple steps back. To make up for this, Zhang and Lapata (2014) combine it with other neural

networks that make the architecture quite complicated.

A third system using neural networks is Wang et al. (2016b). This system uses a planning-based recurrent neural network. The planning was inspired by the observation that a human poet often makes an outline before writing a poem. The system first generates an outline according to the user's writing intent before creating the poem. The system takes a user's input which can be either a word, a sentence or a whole document, and generates the poem in two stages:

1. **Poem Planning**: In the poem planning stage the input query is transformed into x keywords, where x is the number of lines in the poem, and assigns each of these keywords to a line. Each line now has a sub-topic assigned to it.

2. **Poem Generation**: In the poem generation stage for a given line, the system takes all previous generated text and the keyword belonging to that line as input, and generates the poem sequentially line by line according to the sub-topic (keyword) and all preceding lines.

The poem planning features keyword extraction and keyword expansion. Since the system is able to take a whole document as input, the system has to somehow know what words to extract and use as keywords. The algorithm used to evaluate the importance of words is the TextRank algorithm (Mihalcea and Tarau, 2004). It is a graph-based algorithm where each candidate word is represented by a vertex in the graph and edges are added between two words according to their co-occurrence. However, if the user's input query is too short, keyword expansion is needed. Two different methods are used for the keyword expansion:

1. **RNNLM-based method**: The first method is a recurrent neural network language model (RNNLM). The data set used for this is generated by Wang et al. (2016b) using TextRank described above. For each poem consisting of N lines, the words are first ranked in each line scored by the TextRank in the poem corpus, and the word with the highest score is selected.

2. **Knowledge-based method**: The RNNLM method is only suitable for covering words in the topics covered by the training corpus. To solve this problem, a knowledge-based method is introduced. Examples of these are the results of search engines, lexical databases (e.g. ConceptNet), etc. The goal is simply to find words best describing the input words. In this system, encyclopedia entries are used as the source of knowledge.

The poem generation part uses the same encoder-decoder structure with GRUs as in Wang et al. (2016a) and Bahdanau et al. (2014), but slightly modified to support multiple sequences as input. The sub-topic is encoded to a sequence of hidden states, and the preceding text into a bi-directional GRU model. The decoder maintains an internal status vector, and for each generation step, the most probable output is based on the internal status vector, a context vector and the preceding lines.

Lastly, Ghazvininejad et al. (2016) propose a system that creates poetry given a specific topic. The name of the system is Hafez. The system starts by selecting a large vocabulary, and computes stress patterns for each word. The system generates solely iambic pentameter poetry. The CMUdict is used to determine the stressed and unstressed syllables.

Then, given a user-supplied topic, a large set of related words are retrieved. This is done using *word2vec* (Mikolov et al., 2013) by training a continuous-bag-of-words model with a window size 8 and word vector dimension 200. Ghazvininejad et al. (2016) report that the training corpus has a crucial effect of the performance of the model. In this case, the word2vec model was trained on the English Gigaword corpus, a song lyric corpus, and the first billion characters from Wikipedia.

Next, rhyme words are found and put at the end of each line so that they match each other and thus make a rhyme. Also using CMUdict at this stage, the system tries to find rhyming words with related words, however, this is not always successful. The solution is to add fixed pairs of often used words such as *go — know*, which makes the system able to find rhymes in rare topics.

A Finite-state-acceptor (FSA) is built, with a path for every conceivable sequence of vocabulary words that obeys formal rhythm constraints. In total 10^{229} paths are possible, however, most of these have a bad score such as *go we did over we*. In the very end, a path through the FSA is selected, using a RNN for scoring the final outcome. The RNN uses a two layer recurrent neural network with long short-term memory, trained on a corpus of *94,882* English songs. When decoding with the RNN, a beam search is employed. Instead of taking only the most probable word, the N most probable words are taken into consideration. This helps the system find a suitable path through the FSA.

3.2. State-of-the-art image classification

The field of image object recognition has seen various different approaches, however, the main focus will be on the convolutional neural networks because

their results are unmatched in the state-of-the-art image classification. The main focus will be on large scale image classification, as that is what is used in this Thesis. The reasons for this are twofold:

- Performance. In order to generate poetry related to a picture, the first step of recognizing the objects in the picture is important. If the first step is wrong, the rest of the system is based on wrongfulness, and thus the system will not perform.
- Availability. TensorFlow allows any developer to use Inception-v3 which is a high performing image classification system. Therefore using that system is highly desirable.

All of these systems are built for the ILSVRC (ImageNet Large Visual Recognition Challenge)[1]of classifying over 1M images into 1,000 classes.

An early attempt at using CNNs for large scale image recognition was by Krizhevsky et al. (2012). It is a large, deep convolutional neural network with 60 million parameters, 650,000 neurons consisting of five convolutional layers, some of which are followed by max-pooling layers. At the very end of the network are three fully connected layers with a final 1000-way softmax classification.

The first convolutional layer in the architecture filters the *224 x 224 x 3* input images through a filter with 96 kernels having an input size of *11 x 11 x 3*. Max pooling is applied to the output of this layer and becomes the input of the second convolutional layer. The second convolutional layer filters this input with 256 kernels of window size of *5 x 5 x 48*. Max pooling is now performed for the last time. The third convolutional layer has 384 kernels of size *3 x 3 x 256* connected to the normalized and pooled outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size *3 x 3 x 192*, and the fifth convolutional layer has 256 kernels of size 3 x 3 x 192. All of the three fully connected layers have 4096 neurons.

One of the biggest problems of Krizhevsky et al. (2012) is the training time, therefore the system takes advantage of the ReLU Nonlinearity activation function, which is described in Table 2.1. The result is 25% faster training on the CIFAR-10 dataset. In addition, the system is trained on two GTX 580 GPUs with 3GB of memory instead of one, which also drastically increases the computational power available.

The network proposed by Zeiler and Fergus (2014) is based on Krizhevsky et al. (2012), with a couple of improvements implemented. Firstly the window size of the first filter layer was reduced from *11 x 11* to *7 x 7* and made the

[1]http://image-net.org/

stride of the convolution 2 rather than 4. The result is that the architecture is able to retain more information in the first and second layer. Secondly, since the training is only done on one GPU, there is no need to split the data in two in the third, fourth, and fifth layers. Dense connections in these layers are therefore possible. Thirdly, in the pre-processing of the images, the mean of each pixel value is subtracted. This results in each picture being more unique and better results are achieved. Furthermore, the initialization of parameters, batch size and dropout rate are not equal, but it is hard to say how much this influences the results.

OverFeat, Sermanet et al. (2013) propose two different architectures, a fast model and an accurate model. Here I will only talk about the accurate model. The model is first built for classification, expanded into localization and detection. The system is based on Krizhevsky et al. (2012)'s architecture and various improvements are proposed. The size of the network is nine layers where the six first are convolutional layers using ReLU and max-pooling like described above, and the three last are fully connected layers. However, some improvements are made:

- There is no contrast normalization used
- Pooling are non-overlapping
- The first and second layer feature maps are larger and the stride is smaller (size two)

In Krizhevsky et al. (2012), multi-view voting is used to boost performance: a fixed set of 10 views around the image is averaged. A problem with this approach is that it might ignore regions of the image and is redundant when views overlap. Sermanet et al. (2013) explore the entire image by densely running the network at each location and at multiple scales. Therefore more views are available for voting, which increases the robustness of the system. A problem arising with this approach is that the sub-sampling ratio of x36 used above decreases performance compared to the 10-view scheme, because the network windows are not aligned with the objects in the images. To circumvent the problem the last sub-sampling operation is applied at every offset. This removes the loss and yields a sub-sampling ratio of x12 instead of x36.

He et al. (2014) realized the problem of CNN needing a fixed input size, for example the *256 x 256* pixels in an image. This limits both the aspect ratio and the scale of the input image. When applied to images of other sizes, either cropping or wrapping is necessary. These are not desired workarounds,

however. Here a spatial pyramid pooling (SPP) layer is added on top of the last convolutional layer to remove the fixed size input constraint.

The SPP layer pools the features from the last convolutional layer and generates fixed length outputs that are then fed into the fully connected layers. One way of looking at this is adding a completely new layer in the middle of the network. Spatial pyramid matching (Lazebnik et al., 2006) is an extension of the Bag-of-Words model. It partitions the image into divisions from finer to coarser levels, and aggregates local features from them. SPPs were very popular before CNNs. When the CNNs became the superior algorithm to use, SPPs stopped being used. However, they can also be combined with CNNs. SPP has three major advantages when combined with CNNs:

- SPP is able to generate a fixed length out put no matter the size of the input. Meanwhile the window sliding process can be left unaltered.
- SPP uses multi-level spatial bins, while the sliding window uses a single window size. It makes the system more robust to object deformations.
- SPP is able to pool information at variable scales because of the flexibility of the input size.

In theory it does not matter how the rest of the system architecture is, because He et al. (2014) focus on the performance of the SPP layer, therefore they tested this feature in multiple state-of-the-art image classification systems. With their own built network they managed to place third in the ILSVRC 2014 using this technique.

Another remarkable CNN in image classification is the VGGNet (Simonyan and Zisserman, 2014). At this point in time, larger networks are getting researched and multiple architectures of the net are presented, ranging from 11 weighted layers to 19 weighted layers. The only thing in common for these architectures is that the three last layers are always fully connected layers with a 1000-way softmax classifier. The width of the layers (number of channels) is fairly small, starting at 64 in the first layer, and gets multiplied by 2 for every next layer up to 512 in the last layers.

The reason such a big architecture is possible is that the size of the convolutional layers are *3 x 3*. A stack of two *3 x 3* layers effectively gives a *5 x 5* receptive field, and a stack of three layers gives *7 x 7*. So what is the advantage of using three layers of *3 x 3* instead of a *7 x 7*? Firstly, three rectifier layers instead of one makes the decision function more discriminative. Secondly, since $3 * 3 * 3 = 27$ and $7 * 7 = 49$ the number of parameters is substantially decreased. These properties make the system outperform

earlier versions of CNN systems. The architecture of the networks proposed by Simonyan and Zisserman (2014) is now the most widely used architecture for image classification, an architecture which Inception-v3, among others, has adopted.

He et al. (2015) propose a new activation function called Parametric Rectified Linear Unit (PReLU), and is the first system managing to surpass human level performance on the ILSVRC visual recognition challenge. This is done in two ways:

- PReLU generalizes the traditional rectifier unit (ReLU). It improves model fitting with almost no additional computational cost and little risk to overfit.
- A new robust initialization method is proposed which considers rectifier non-linearities.

Another result from these contributions is that they allow for training wider and deeper networks, resulting in better performance.

The difference between PReLU and ReLU is not big. Both functions give the same output, namely $f(x) = x$, when $x > 0$. However, when $x < 0$, ReLU returns $f(x) = 0$. PReLU on the other hand returns $f(x) = ay$, where a is a coefficient adaptively learned when training.

Most CNNs are initialized by random weights drawn from Gaussian distributions. A problem with this is that very deep models have problems converging. One solution to this is *Xavier* initialization, but even that is sub-optimal as it is built for linear activation functions. A new way of instantiation is proposed based on the idea of investigating the variance of the responses in each layer.

The core architecture of He et al. (2015) is similar to the architecture in the VGGNet, with a couple of modifications. Firstly, the filter for the first layer is *7 x 7* with a stride of 2. Secondly, the other three convolutional layers on the two largest feature maps are moved to the smaller feature maps. Thirdly, SPP is applied before the first fully connected layer.

Ioffe and Szegedy (2015) introduce batch normalization. The goal of this technique is to reduce Internal Covariate Shift (ICS). ICS is the change in the distributions of internal nodes of a deep network when training. ICS *"dramatically accelerates the training of deep neural nets"* according to Ioffe and Szegedy (2015). In addition to this, it also has a beneficial effect on the gradient flow through the network, by reducing the dependence of gradients on the scale of the parameters or their initial values. The result of this is training with higher learning rates without divergence. A last positive side is that it regularizes the model and reduces the need for dropout.

Batch Normalization was applied to the best performing ImageNet classification network, and was matching its performance using only 7% of the training steps.

Lastly, Szegedy et al. (2016) proposed the Inception-v3 architecture which is used for this Thesis. The architecture it explained in depth in Chapter 5.

4. Data set

This chapter covers the data set produced and used in this project. Existing data sets of the right size and content are not available due to various copyright protections, necessitating the gathering of a new data set.

A data set in this context is a collection of data used to train the Artificial Neural Network created in this Thesis. Optimally, a data set should consist of data as close as possible to the desired results of the system, because the goal of training computer models is to mimic the data set as closely as possible. However, collecting a data set consisting of poetry is very inconsistent, meaning that every poet writes differently and uses different words and expressions. The result is that the available sample of reasonably consistent poetry is rather small. In comparison, hundreds of thousands of song lyrics — essentially rhythmic poems — are readily available. Hence, those in the field of poetry generation commonly use song lyrics rather than poetry as their data sets. Therefore, just like Ghazvininejad et al. (2016), song lyrics were chosen as a base for the data set in this Thesis.

Due to copyright protections, the data set cannot be distributed further. All songs are copyrighted property of their owner, and therefore cannot be reproduced or fully quoted in this Thesis either.

4.1. Gathering the data set

The data set was collected from *www.mldb.org*, an online song lyrics database. A Python script was written to connect to their site, sort through the HTML files of the site and find the song text, artist and album. Beautiful Soup[1], which is a HTML and XML handling library, handles the HTML-document collected. Beautiful Soup handles the HTML file by making a parsing tree, making it easy to navigate and handle the data provided in the HTML file.

Figure 4.1 shows the tree structured layout of the site, and therefore the tree structure to navigate in the Python script. The main page is shown at the top. One step down is the alphabetic order of all the artists available on the site. Further down the tree are the artists for the letter chosen in the

[1]https://www.crummy.com/software/BeautifulSoup/

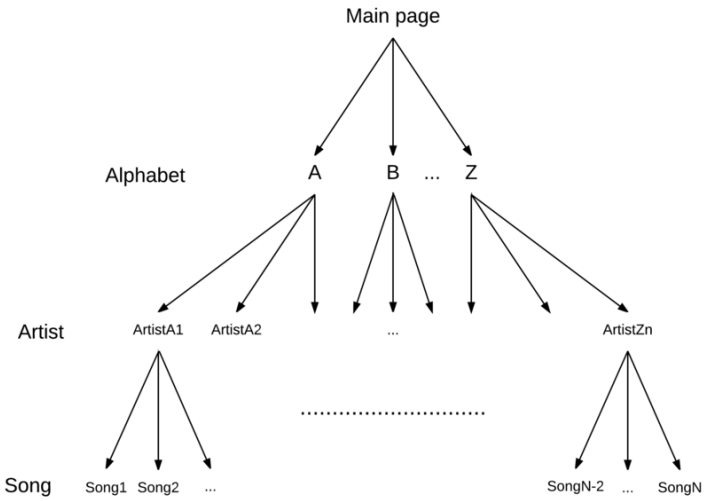


Figure 4.1.: The tree structure of *www.mldb.org*

step over. Beneath the artists are the songs. The song text, artist and album are then gathered from the leaf nodes.

One problem when running scripts that crawl a website is that of getting IP-banned. *www.mldb.org* only allows for 300 song visits a day. One way of overcoming this is to change IP-address every 250 songs visited. By doing this, the compilation of data advances faster and more experimenting can be done.

4.2. Preprocessing of the data

Now when the data is collected the total number of songs in the set is 206,150, with a vocabulary, or the total number of unique words, of 391,363 and a total of 46,346,930 word tokens. The problem now becomes that many songs are not eligible to make it into the final data set. The biggest problems with most songs are these:

- **Language**. Between one third to half of the songs gathered are not in English, and must be removed. This is also why the vocabulary is almost 400,000 unique words. This is easily done by using NLTK's built in English vocabulary. However, some songs are written primarily in English, but also contain words from other languages. The solution for this is to say that 95% of the words in a given song must be English words.

- **Escape characters** are characters with a special meaning in a document such as newline, backslash or the quote character. When reading the HTML file using Beautiful Soup, these appears as they are read from the text. Removing them is done by using regular expressions.

- **Custom made messages by the sender**. The custom messages such as *Submitted by x* and *Thanks to y for providing these lyrics* are difficult to find and remove. However, by physically looking for them, it is possible to remove a big portion of these messages. The data set still contains them, but they are very rare and do not seem to have any influence on the results.

- **Verse and chorus notation**. This category consist of lines such as *[Chorus]* and **Verse 4**. One way of attacking these lines is to cut away any lines containing * or *[]*. Also removing *Chorus* if it is on its own line.

- **Name of artists**. If multiple artists are performing the lyrics together, the lyrics often state which artist is performing a given verse. Removing the name of the artist from the verse is a solution, however, it does not fully remove all occurrences of artist names.

- **Notation of physical movements**. In some cases, phrases such as **Inhales Deeply** are present in the text. Again the solution is to remove the lines containing *.

After these issues are taken care of, the data set consists mostly of text that is desirable for the recurrent neural network to learn. The total amount of songs in this text is 80,608. The vocabulary is 91,097. This amount is too large for multiple reasons: Firstly, the training time of the LSTM network increases because of the need of having the whole vocabulary as a list. Secondly, when running the system, many words will be predicted as a zero percent chance of being the next word. Thirdly, search time when finding the optimal path for the tree is increased. The gain is that you have a more diverse word base to predict from. However, as mentioned, in a vocabulary as big as 90,197, most words just return a prediction of *0.0*, rendering the use of a big vocabulary useless. Basically, you gain little to nothing by having a big vocabulary, but the drawbacks are clear.

Another dimension regarding the size of the data set is the fraction of related words covered by it. If the data set cannot cover the related words, the system will under-perform. Therefore, choosing a data set big enough to cover related words is important, while making it small enough to not waste

time and resources. A small experiment was conducted to see how well certain vocabulary sizes covered the related words. Five random pictures were chosen, with five different vocabulary sizes. The vocabulary size considers the most frequent words, i.e. the data set using the 15,000 word vocabulary is the set of songs containing only words that are among the 15,000 most frequent words in the total data set, all other songs are dismissed. The results are shown in Table 4.1.

Table 4.1.: Vocabulary coverage. The top row represents the vocabulary size, while the leftmost column represents each picture.

Picture	15,000	20,000	25,000	30,000	35,000
Zebra picture	30.0%	31.8%	35.5%	42.7%	43.6%
Pool table picture	53.6%	58.6%	61.2%	63.3%	66.1%
Radiator picture	32.1%	42.5%	46.3%	50.7%	53.0%
Vacuum cleaner picture	37.3%	40.5%	43.7%	53.2%	55.0%
Desk picture	35.1%	48.5%	51.1%	53.2%	55.1%

This experiment has a very small sample size, and is inconclusive, but the tendency is that the related words are covered at an acceptable percentage level when the vocabulary size reaches 30,000, which points towards the need of the vocabulary having at least that size. The percentage of covered words do not rise much past 30,000. Since no other reason for having a larger vocabulary is found, all songs containing words that are not in the top 30,000 vocabulary are removed, and this becomes the data set to do the LSTM network training on. The final size of the vocabulary for this data set becomes 27,601.

4.3. Final data sets

Three different data sets and their properties are displayed in Table 4.2.

Table 4.2.: Data sets properties.

Data set	Vocabulary size	Number of songs	Total number of tokens
Data set 1	91,097	80,608	18,807,460
Data set 2	27,061	40,685	8,712,213
Data set 3	14,725	23,697	4,868,042

Data set 1 is the full data set pre-processed, while data set 2 is the data

set where the top 30,000 vocabulary is included, and data set 3 is the data set where the top 15,000 vocabulary is included.

Data set 2 is used to train the LSTM network. The training process is described in Chapter 6.

5. Architecture

The architecture for the system implemented is introduced in this chapter. The first step after obtaining an image is to run it through Inception to classify the image. The output from Inception is the five top scoring classes. If these classes are scored higher than a set threshold, they will be taken further into the system. If the score is beneath the threshold, the result is discarded. The results from Inception are then used to find related words. This is done in two steps using ConceptNet. Firstly, ConceptNet's related words feature is used. A list of related words is returned, all scored. Secondly, the core of ConceptNet is used. Edges are found for the different concepts, and the concept at the other end of the edge is saved and scored. When all the related words are found, rhyme words are explored. The goal is that both rhyming words should be related to the picture, but it is not always doable. Higher scoring related words will be prioritized during the exploration of rhyming words.

The next step is to find the optimal path through a tree structure to construct a sentence. The path starts from the start of the sentence and ends on the rhyme word at the given line. Any number of lines can be generated. When a line is generated, an attempt to check the grammatical structure is made, but its use is limited. When a desirable number of lines have been generated, the system is done and the poetry is generated. Figure 5.1 shows all steps from an input image to the generated poetry. The data set gathering is not included in this figure.

5.1. Inception

Inception-v3 Szegedy et al. (2016) is the object recognition part chosen for this thesis. The network is a convolutional neural network (CNN) as described in Section 2.3. It is available through TensorFlow in both Python and C++.

The architecture of Inception-v3 is large, consisting of a total of 42 layers. The outline of the network is as follows: The input image is cropped to a size of *299 x 299 x 3*. The first three layers are *3 x 3* convolutions and are used to capture high-level features present in the image. These layers

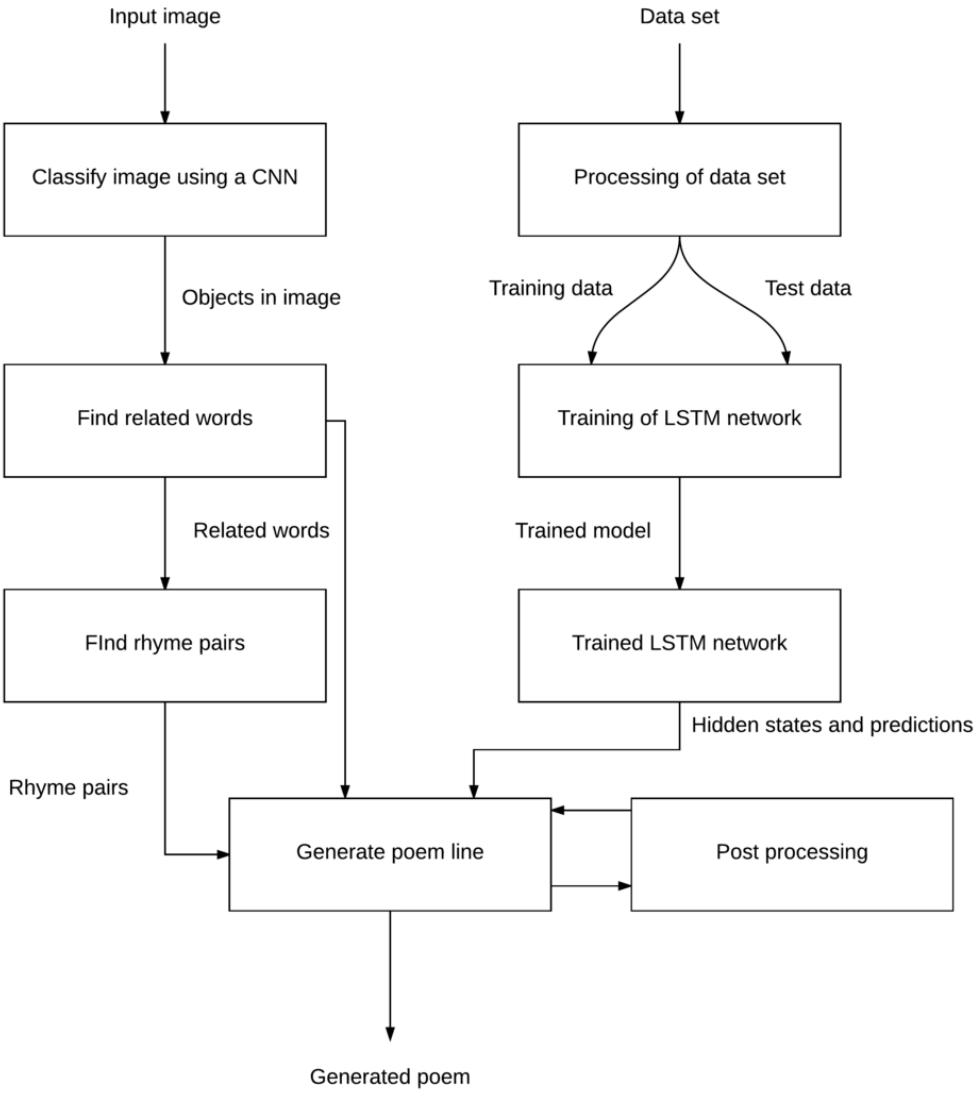


Figure 5.1.: A high level overview of the system.

are followed by a pooling layer, thereafter several more convolution layers. Now the inception modules are introduced into the network. An inception module is simply a replacement of a bigger convolution into multiple smaller convolutions, where the result is to capture more and better features. After the convolution layers, 10 inception layers are following. These vary from *35 x 35* to 8 x 8 in size. At the end of the network, there is a pooling layer, followed by a linear layer. The last layer is the softmax classifier.

Inception-v3 is pre-trained on the 2012 ImageNet Large Visual Recognition Challenge (ILVRC). The training was done by stochastic gradient using TensorFlow. Each of the 50 replicas was running on a NVidia Kepler GPU with batch size 32 for 100 epochs. Momentum is used with a decay of 0.9. The learning rate was 0.045 decayed every two epochs using an exponential rate of 0.94. In addition, gradient clipping with threshold 2.0 was used to stabilize the training.

In the ILVRC, the task is to classify a picture into one of 1,000 classes. The classes have no particular themes, however, 400 of the classes are animals. The rest of the classes are quite diverse. It can be anything from *CD player* to *castle*. A consequence of using Inception-v3 in this Thesis' system though, is that a restriction is set to make sure the performance is not hindered by the first step of the architecture: The images used in the experiments must contain at least one object recognizable by Inception-v3.

5.2. Finding related words and rhyme pairs

The keywords returned by the CNN are used to gather a large set of related words. All results over a certain score are taken into account. This set score is 0.10. Using ConceptNet, the quality of the related words is mixed. For instance, the relation between *car* and *gasoline* is only *0.30* out of 1. ConceptNet also returns some slightly unrelated words. For instance, obtaining related words on the term *zebra* returns the word *only*.

The related words process is conducted as the following: First the related words feature in ConceptNet is used. Given a set of key words from the CNN, the call returns a number of related words along with a similarity score between 0 and 1. The next step concerns the edges of the concepts belonging to the keywords. Each edge has another concept end point, and these are retrieved along with a score. The score is now between 1 and 12. Therefore the score is divided by 12, to normalize it. The scores from related words, and nearby concepts are treated equally. If the system has few high scoring related concepts, the system will prune the highest scores and retrieve concepts related to them. This repeats until the system has at

least 200 related words. All related words need to be in the vocabulary, so the LSTM network can use its predictions to predict these related words.

When the related words are found, the system looks for rhyme pairs. This is done using the CMUdict, which finds the pronunciation of a given word. For instance, the word *dictionary* is returned as *D IH1 K SH AH0 N EH2 R IY0*. Here, the numbers are the syllable stress, and the letters are the pronunciation. The two rules of rhyming in Section 2.1 are used to find the rhyming words. The first check is done to see if the endings of the words are pronounced the same, i.e. the ending of both the CMUdict objects are equal. The length of the rhyme does not matter, though it must be at least one syllable long for each of the rhyming words. In the *dictionary* example, this means that the rhyming word must at least end on *R IY*, since that is what belongs to the last syllable. The second rule states that the consonant sounds preceding the rhyme must be different. In this case, the consonant sound before the rhyme is the *N* before *EH2 R IY0*, so the rhyming word cannot have *N* preceding the *EH2 R IY0*. If the word satisfies these constraints, a rhyming word is found.

A consequence of this is that slant rhymes are not allowed in the system. Therefore, mostly perfect rhymes are present in the rhymes. This is a design choice, because slant rhymes such as *milk - talk* do not sound particularly good. However, if the system cannot find any suitable rhyming words, the second rule of rhyming is discarded, and only the endings of the words are checked.

Ideally, both rhyme words should be related words to the initial image. If there are no related words rhyming with each other, the system looks for other words rhyming on a related word in the vocabulary. The highest scoring related words and their rhyming words are preferred. Now the related words and rhyming pairs are found.

5.3. The Long Short-Term Memory (LSTM) network

The task of the LSTM network is to predict the next word in a given sequence. It takes a word and the previous hidden state as an input, before producing an array of the scores for each other word in the vocabulary. It then updates the hidden state taking the whole sequence of previous words into consideration. Many different types of architecture were tried, before choosing the best performing one. The training of the different networks is described in Section 6.1.

Before training the network the data set is pre-processed to fit into the network. Firstly the data set is split into three sets: training set, validation set and test set. The training set is the part of the data set the network is training on. The validation set is a data set used to evaluate the training and is not used for training. Finally, the test set is the set used to test the network at the end of training, when the network has not yet seen the test set. The training set is chosen to be 80% of the data set, while both the validation and test sets are 10%. The purpose of this splitting is to attempt to make sure that the LSTM network does not overfit on a specific training set. Splitting it up and measuring the word perplexity on the validation and test sets in addition to the training set tests the network on data sets it has not been trained on, and is a good indicator of if the network is overfitting on the training set.

After the data set is split, the vocabulary is built. All the words in the data set are tokenized and given a unique ID to represent them. The IDs are frequency based, meaning that the IDs are given a higher ID number the less frequently they are used. For instance, ID *0* is the most frequent word, ID 1 is the second most frequent word, and so on. The data sets are then converted into a file of IDs representing the original data set files.

The model embeds the IDs into a dense Vector Space Model (VSM) representation before feeding it further into the architecture. This allows the model to more efficiently represent knowledge about the words in the data set. The embedding matrix is initialized randomly, but as the training goes on, the model learns to differentiate between words by looking at the data set.

The third and final processing step is to take batches of the data set and convert them into tensors that TensorFlow can use to train the model. The reason for only training in batches and not on the whole data set is to save memory when training. Otherwise it becomes a limiting factor regarding the size of the network. Furthermore, it makes the training go faster because after each batch the weights are updated, it is not needed to go through the whole data set before updating the weights.

When the pre-processing is done, the data set can be fed into the neural network. Several different architectures were tried, and the best performing one is described here. The model has four layers: One input layer, used to represent the words coming in from the data set. This layer is followed by two hidden layers with the size of 1100 cells. The last layer is a softmax layer used to give the predictions for the next words. The core of the network is the LSTM cells. These are described in Section 2.3. The LSTM cells are found in the two hidden layers, and compute the possible values for the next

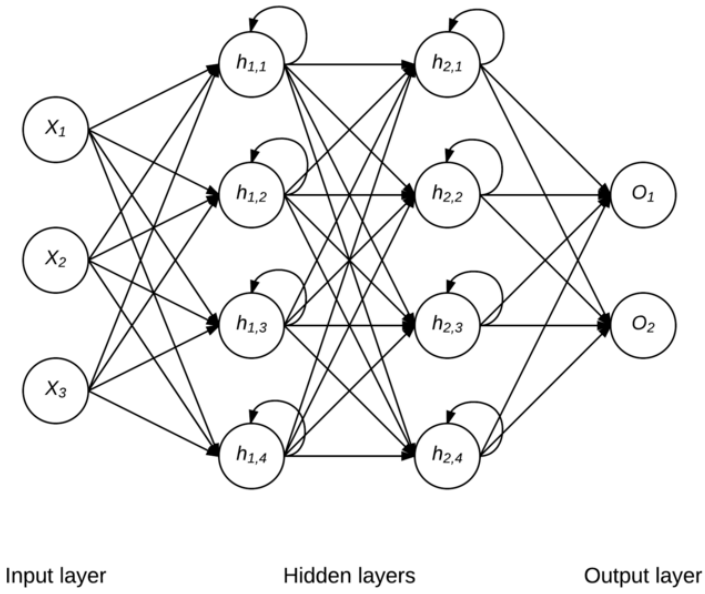


Figure 5.2.: Architecture of the LSTM network.

predictions. Figure 5.2 shows the architecture of the network.

The loss function the network tries to minimize is the average negative log probability of the target word:

$$loss = -\frac{1}{N}\sum_{i=1}^{N} ln\ p_{\text{target}_i} \tag{5.1}$$

In Equation 5.1, N is the total training set, *target* is the target word, and i is the word being looked at. The log loss is the cross entropy between the distribution of the true labels and the predictions given by the network. Therefore, by minimizing the log loss, the weights are optimized to give out a precise distribution of true labels. It is the equivalent to maximizing the product of the predicted probability under the model.

This introduces the measurement of how well this LSTM network is performing during training: per-word perplexity as shown in Equation 5.2.

$$e^{-\frac{1}{N}\sum_{i=1}^{N} lnp_{\text{target}_i}} = e^{loss} \tag{5.2}$$

The word perplexity is a measurement of how well a probability distribution can predict a sample. The lower the perplexity, the more correct the predictions are for the next word, or in other words, the less confused the

model is about the next word. When training the model, a lower perplexity is almost always to prefer. However, in some cases it can be very misleading, especially in the case of song lyrics. More in depth explanation of why this is, the results of the training, and the parameters used to train, are described in Section 6.1.

The optimizer used in the model is the gradient descent optimizer and it is used along with backpropagation to minimize the loss function during training. The backpropagation algorithm uses the error values found by the loss function, L, to calculate the gradient of the loss function with respect to the weights, w, in the network, $\frac{\partial L}{\partial w}$.

This gradient is fed to the optimizer which updates the weights in an attempt to minimize the loss function. The optimizer used in this network, gradient descent, is a first-order iterative optimization algorithm. The basic idea behind finding the minimum of a function using gradient descent, is to take steps proportional to the negative of the gradient. However, getting stuck in a local minimum can cause problems. To overcome this, the learning rate is set so that the steps taken search a larger search space before decreasing and attempting to find the global minimum.

The network description is now finished. However, by introducing dropout (Srivastava et al., 2014), the network can be further optimized. Dropout is a technique where randomly selected neurons are ignored during training. Each run, every neuron is given a chance to drop out, e.g. 20%. The result of this is that their contribution to the forward pass and weight updates in the backward pass is not applied. Generally, as a network learns, neuron weights find their own contexts within the net. Neighboring units start to rely on the close by weights which makes the model more fragile and too specialized on the training set. With dropout, neurons become less reliant on neighbors and better at generalizing on the training set, thus less likely to overfit on the training set.

The workflow when the network is trained is as follows: The network takes a word and a hidden state as input. This word is then transformed into a dense vector representation learned while training. The word is then run through the network, creating a vector of the predictions of the next word and a hidden state update where all previous words are taken into account. For instance, when the word *zebra* is run through the network as the first word w_0, the input is an initial state h_0 and w_0. The output is the predictions for the next word p_0 and a new hidden state h_1. Now a new word, w_1, is chosen from p_0. To get the predictions for the next word, w_1 and h_1 are fed to the net which returns p_1 and w_2.

5.4. Poetry generation process

Now that the LSTM network has been described, the poetry generation process can be introduced. The poetry generation takes the related words, rhyme pairs and a trained LSTM network as input. The next step creates a tree structure to find the highest scoring paths through the tree, before POS-tagging is applied to the paths, and returns the most optimal path that fits the grammar constraints.

The first step of the poetry generation process is to generate a tree for each line in the stanza. One node symbolizes one word and its syllables, and each edge is the score between one node and another. This way the system can check if the next word after *I don't* should be *like* or *browser*. The score between two nodes is provided by the trained LSTM network.

The root of the tree is an empty string with the initialized hidden state. Based on the root and the hidden state, the system takes the top 30 words predicted and sets them as the root's children. For these 30 words predicted, the system generates 30 predictions. This process is done when the syllable constraints are fulfilled. Figure 5.3 shows the tree structure of this idea.

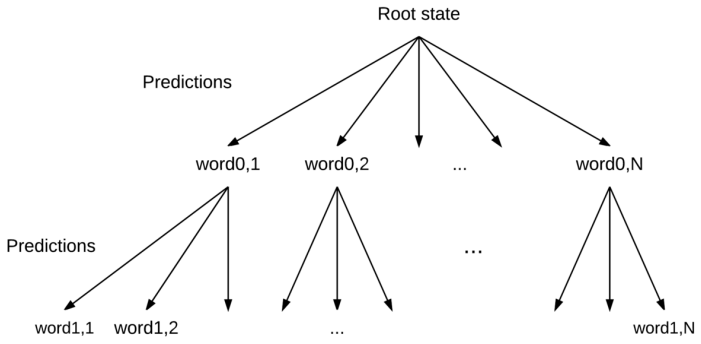


Figure 5.3.: The tree structure of the system's path finding.

However, this tree is too large to be effective and the searching takes too long. In the worst case scenario, when all words are monosyllabic, a tree finding the path to an eight syllable length phrase will consist of 30^8 nodes. Due to various calculations done in TensorFlow, only about five nodes can be looked at per second. Therefore, searching through that many nodes is unacceptable. In addition, having that many nodes in memory is not possible. The solution to this is two folded: Firstly, perform a depth first search while deleting all child nodes that have already been looked at. Secondly, prune the tree based on the score of a line or syllabic constraints.

The depth first search takes the syllable constraints into account when running. If the only syllable constraint is to match a certain length of syllables, the search finds a path to a leaf node containing words that gives the desired syllable length. When the depth first search reaches the leaf node, the system generates a score of how good the line is, which is the sum of each prediction for each word. The next step is to see how good the prediction to the different rhyme words is. Based on how relevant the word is to the image, and how well the line is scored to different rhyme words tuples, one tuple is chosen. This tuple contains the rhyme word for this line and the rhyme word for another line. This line can be positioned anywhere in the stanza, depending on the rhyming scheme chosen for the stanza.

Now, when the leaf has been looked at, the tree search goes back to look at the leaf node's parent, sets the current node to be looked at as one of the siblings of this node, and calculates the next predictions for this new node. This is done 30 times, once for each of the 30 top predictions. When all the siblings of a node have been looked at, the siblings and the node are deleted. The current node being looked at is now one of the siblings of the deleted node's parent. This way, no more than $30 * 8$ nodes are kept in memory, speeding up the process of finding a suitable path.

Pruning of the tree is done in two ways: Based on the score, or based on different syllabic constraints. At any given point in time of the run, n number of top scores are kept of the best performing lines this far. A node is skipped if does not fulfil the following constraint:

$$node_{\text{score}} > \frac{S_{\text{node}}}{S_{\text{tot}}} \times \frac{top_score[-1]}{2} \tag{5.3}$$

S_{node} is the syllable count for a given node, S_{tot} is the total number of syllables needed to finish the line. *top_score[-1]* is the worst top score achieved at this point. By using a simple heuristic, many sub-optimal paths are dismissed, and each run can finish in about an hour.

The score of a given line is calculated in two phases: One part is from the words generated in the sentence, as described earlier, and the prediction to the rhyming word. Equation 5.4 shows the formula.

$$score_{\text{line}} = \sum_{i=0}^{N} pred_i + pred_{\text{rhyme}} \tag{5.4}$$

$pred_i$ is the prediction score for the i'th word, N is the total number of words in the line, and $pred_{\text{rhyme}}$ is the prediction score for the rhyming word. One important fact to note is that these values are continuously normalized against other values in the top score list, therefore the line score and the

rhyme word prediction are evaluated as equal parts. If this was not the case, the line score would almost always have a higher score, because it is the sum of multiple predictions, while the rhyme word prediction is only one prediction.

The system tends to repeat words, since the network is trained on a song lyrics data set. Song lyrics often repeat themselves, phrases such as *I love you <eos> I love you <eos> ...* are very common. <eos> is the end of sentence symbol. Therefore words already used in a line receive lower scores, and are avoided if possible. The same word will still occasionally appear multiple times, but rarely.

In addition to the system repeating words, the system also prefers to use easy and safe words. Because of this a higher score is given to the related words. This enhances the performance of the system, and forces the system into exploring new words it otherwise would not have.

A problem the system cannot handle very well is the transition between the generated line and the rhyme word. POS-tagging is added to address this problem. The POS-tagger first tags the line generated, and returns the tags for the line. It then tags the rhyme word separately. However, tagging a word without any context is not possible, therefore the Brown corpus is used to determine the tag of the rhyme word. The Brown corpus is a 1M word corpus already POS-tagged. Using the frequency of the POS-tagging for the rhyme word present in the Brown corpus, it is possible to see what the rhyme word is most often POS-tagged as. Now, by fixing the rhyme word at the end of the line, and POS-tagging again, this time the line and the rhyme word together, the rhyme word gets another POS-tag. If this new POS-tag matches the POS-tag from the most used POS-tag in the Brown corpus, the line is accepted.

At this point, one line of poetry is generated. The next step is to set the root to the hidden state representing the line with the highest score, so that the hidden state represents this line. Since the root now is not an initial state, but rather a state representing the line generated, the system takes the generated line into account when predicting future words. This continues as the stanza is being generated.

Finally, it is worth mentioning that the system is able to support any sequence of syllables given, e.g. *1101101*, and find the best scoring path where each line contains exactly this syllable pattern. It can also generate any amount of lines, and any syllable length.

6. Experiments and Results

In this chapter, the experiments conducted are reported. The first experiment is the training of the LSTM network to try to minimize the word perplexity of the model. The second and third experiments are evaluating the quality of the poetry generated.

6.1. Training of the LSTM network

The first experiment was conducted to decide the architecture of the LSTM network. When training a network, many different parameters have to be set. Most parameters are chosen by replicating Zaremba et al. (2014). The most important parameters are:

- Batch size
- Learning rate and learning rate decay
- Probability of keeping a neuron (dropout)
- Number of steps unrolled in the LSTM layers
- Initializing of parameters

When training, the learning rate starts at 1.0, and after 14 epochs the learning rate is decreased by a factor of 1.15 after each epoch. The batch size is 20, and the parameters are uniformly initialized at a range of [-0.04, 0.04]. The dropout rate is 65% for non-recurrent units. The number of steps unrolled in the LSTM layers is 35. The training time for the best performing network was 34 hours and 41 minutes on a NVIDIA GeForce GTX 970 with 4 GB memory.

To see how the word perplexity changes when changing the architecture, the following parameters were tested:

- Number of layers
- Number of hidden units
- Number of epochs

Table 6.1.: Word perplexity for different architectures.

Number of layers	Number of hidden units	Epochs ran	Test data perplexity
1	500	35	124.3
1	500	53	117.4
1	800	35	115.3
1	800	53	112.0
1	1100	35	89.9
1	1100	53	65.0
2	500	35	86.3
2	500	53	57.0
2	800	35	64.0
2	800	53	41.6
2	1100	35	38.9
2	1100	53	36.7

Table 6.1 shows the word perplexity of different architectures. The lower the word perplexity, the less confused the network is.

It is quite clear from the table that the larger the network is, the better the results are. The reason for not trying bigger networks is due to memory constraints on the NVIDIA GeForce GTX 970 the network is trained on.

Getting this low perplexity is quite remarkable. Zaremba et al. (2014) managed to get a word perplexity of *68.7* on the Penn Tree Bank data set, but that is with a vocabulary size of only 10,000, while this vocabulary size is 27,601. Usually when you train with a bigger vocabulary, the network should be more confused, i.e. the word perplexity should increase. The explanation is quite simple, however: The reason is the data set. Song lyrics tend to use a small variety of words, and often repeat lines. The network tries to mimic this behaviour. The result is that the predictions of rare words are often *0.0*. This has big implications on the performance of the system.

What does the perplexity say about the model? Recall that word-level perplexity is a measure of how close a probability distribution is to another probability distribution, **on average per word**. With a vocabulary of size X, and e.g. half of the predictions return *0.0*. The probability distribution will always guess half of it right because it is always *0.0*. A vocabulary bigger than vocabulary X will have a bigger fraction of its predictions return 0.0 because of the song lyrics properties of repeating words and use different words rarely. Therefore, when the vocabulary grows, it decreases the word perplexity, by a considerable amount. A word perplexity of *247.5* was reached with the same architecture as described in Chapter 5 using data set 3 (with a 14,725 token vocabulary), while on data set 2 (with a 27,061 token vocab-

ulary) it reached *36.7*. Data set 3 does not return *0.0* on half its vocabulary, but a smaller fraction because of the frequency of the words versus the total number of word tokens, due to the properties of song lyrics. Table 6.2 shows an example of this. It shows the number of times a given word appears in the data set. A word such as *Zebra* only appears seven times in data set 2. However, it still appears six times in data set 3, although that data set is only half as large (4.8 M token vs 8.7 M). Data set 2 will more likely return a lower prediction of *zebra*, maybe even zero. Data set 3 has a higher chance of returning a non-zero output and therefore the perplexity increases.

Table 6.2.: The number of times a word appears in the vocabulary, showcasing the affect of some words being heavily favored in the data set.

	I	but	you	zebra	golf	swallow
Data set 2	166,007	42,017	154,207	7	5	223
Data set 3	89,046	23,785	76,843	6	4	153

It is hard to say how good the results are due to different word perplexity results with different vocabularies. However, the best performing architecture from this experiment is chosen.

6.2. Experiments on the generated poetry

Two different experiments are conducted to test the system. In the first experiment human judges choose an image and evaluate each poem on three different measurements: *Meaningfulness*, *Grammaticality* and *Poeticness*, as described in Section 2.4. In the second experiment, human judges are tested to see if they could tell the difference between the poetry generated by the system, and the poetry written by a human. A total of 46 persons participated in the experiments, a total of 153 stanzas were rated on the three criteria, and a total of 38 evaluations were done to decide if the poetry was from a human or a computer.

The stanzas are chosen to have these properties: The only rhythmic constraint is that a line has to be 8 syllables long, i.e. any syllable pattern is allowed, but it has to be 8 syllables in length. The chosen rhyming scheme is *AABB*, and each stanza has to be 4 lines long. One stanza is generated per image.

The experiments are executed as follows: First the participant is asked to find at least three images of his or her choosing. The reasons for this is to

avoid testing the system on images and use the best generated poetry. When the participant has chosen three images, the images are run through the system and the stanza is retrieved. The participant is then asked to rate the stanza. Next, the participant is shown an image and corresponding human generated and machine generated stanza. The participant is then asked what stanza is the human generated one. Finally, the participant is asked what their overall thoughts of the system are.

A highlight of the poems is listed Figure 6.1, while a selection of the rest can be found in appendix A. Ten randomly selected poems are showcased in this section to show a variety of different scored poems. The input image is to the left while the generated poetry belonging to the image is on the right. Under the poetry, the score for the poem is stated, in the order of poeticness, grammaticality, meaningfulness. Unless stated otherwise, the images are used under the license of Common Creative Attribution 2.0. Link to this license is found in appendix A.

6.2.1. Grammaticality, poeticness and meaningfulness

The goal of the first experiment was to evaluate each poem on the three criteria: grammaticality, poeticness and meaningfulness. The average score, standard deviation and median of all the poems are shown in Table 6.3.

Table 6.3.: The average score (AVG), standard deviation (SD) and median (MD) of all poems evaluated.

	Poeticness	Grammaticality	Meaningfulness
AVG	2.614	2.381	2.050
SD	0.519	0.470	0.712
MD	2.8	2.5	2.0

The scores show that poeticness scores higher than grammaticality and meaningfulness. The reason for this is the guarantee of the lines being eight syllables long, and almost every line rhymes. When the system cannot find suitable rhyme words, sub-optimal rhyme words are chosen, such as *lifeboat* and *sailboat*. These are not perfect rhymes. The grammaticality score is a bit lower, and the meaningfulness is slightly above the "partially meaningful" score. Both of these are influenced by the system predicting ending rhyme words using zero-values.

Another interesting point is comparing the results of poems with two or more lines where the rhyme word is predicted by a non-zero number against poems containing zero or one such lines. As mentioned in Section 6.1, the

The sun is in my big raincoats
I don't know what to do scapegoats
I'm raining and it looks like rain
There's so much for me to abstain

Score: [3.0, 2.7, 2.0]

(a) Picture from "Red Umbrella" (CC BY 2.0) by DLG Images. URL: https://www.flickr.com/photos/131260238C%40N08/16722739971/.

I don't know why it feels like crabs
That make me want to look at cabs
So come on lets get out of zoo
And dive into my big canoe

Score: [3.0, 2.5, 1.0]

(b) Picture from "Jellyfish" (CC BY 2.0) by Jennifer C. URL: https://www.flickr.com/photos/29638108%40N06/34440131755/.

I don't want to see a big lens
The way you look at me and pens
Focus in my photography
When flash of your biography

Score: [2.8, 2.6, 3.0]

(c) Picture from "A Source of Stability" (CC BY 2.0) by kyle tsui. URL: https://www.flickr.com/photos/wackyland/4305171790/.

It's so hard to be in love band
I don't know what it is but sand
You make me feel like a man roll
With my head in the wall and coal

Score: [2.0, 1.0, 1.0]

(d) Picture from "snb13088" (CC by 2.0) by zasqwe. URL: https://www.flickr.com/photos/76010111@N06/7220716772/.

I want to be with your tent group
In the shape of your hand and troop
My life is an operation
So come on lets go now station

Score: [2.5, 3.0, 2.5]

(e) Picture from "KFOR 12 training" (CC by 2.0) by The U.S. Army. URL: https://www.flickr.com/photos/soldiersmediacenter/3953834518/.

I don't know what to do cute bear
The way you look at me when fair
And I'm so in love with teddy
You're just a part of already

Score: [2.8, 2.7, 3.0]

(f) Picture by Yumeng Sun with permission to use.

I can see it in potatoes
It's been a long time tomatoes
You and me aren't meant to be red
So come on let's have some fun tread

Score: [2.5, 2.0, 1.3]

(g) Picture from Ole Steinar Skrede with permission to use.

I want to tell you that you're mute
Dance with me all through the night flute
So we can get out of keyboard
It's been a long time harpsichord

Score: [2.8, 2.7, 2.0]

(h) Picture from "Violin Still Life" (CC BY 2.0) by Chris Yeh. URL: https://www.flickr.com/photos/9623061%40N06/32426451122/.

Hole in the back of my bottle
I don't know what it's like throttle
It makes me feel this way grape wine
And when I'm with you flask align

Score: [2.8, 2.5, 2.5]

(i) Picture by "NAPA 2012" (CC by 2.0) by cdorobek. URL: https://www.flickr.com/photos/cdorobek/8093546797/.

The shape of things to come necktie
Cuffs with me like a alibi
Lapels in black and white eyes bow
You know what I mean, just allow

Score: [2.0, 1.4, 1.0]

(j) Picture from Ole Steinar Skrede with permission to use.

Figure 6.1.: A randomly chosen selection of the poems generated.

rhyme word is often predicted by a zero value, in which case the POS-tagging system and the relevance of the word decide the rhyme word and the corresponding line.

There can be two reasons for that a score is zero. The first reason is that the network has been trained on the word, but it does not think it is a good fit and thus return zero. The second reason is that the LSTM network has been trained on a data set containing few occurrences of the rhyming word, and therefore outputs a zero because it does not know if it fits well. If the word is a non-zero value it is guaranteed that the word is represented enough in the data set, however, if it is a good fit is up to the network to decide. Therefore, by looking at the scores of poems where the rhyme words of two or more lines are not predicted as zero, the implications of the LSTM network are easier to evaluate and are correlated to Research Question 2: Are the predictions from an LSTM network enough to generate grammatically correct poetry?

In total 33 of the 153 stanzas contain two or more lines where the rhyme word is predicted by a non zero value.

Table 6.4.: Comparison of poems containing two or more lines where the rhyme word is predicted by a non-zero value, and poems containing one or two such lines.

Number of non-zero prediction lines	Method	Poeticness	Grammaticality	Meaningfulness
≥ 2	AVG	2.793	2.691	2.464
	SD	0.118	0.158	0.640
	MD	2.800	2.700	2.500
< 2	AVG	2.574	2.315	1.962
	SD	0.561	0.488	0.703
	MD	2.8	2.5	2.0

The scores shown in Table 6.4 indicate that when it is known that the system has an opinion of the rhyming word, the performance increases. Note that the opinion does not necessary have to be good. The average poeticness goes from 2.614 to 2.793, while grammaticality increases from 2.315 to 2.691 and poeticness increases from 2.050 to 2.464. However, is there a correlation between these results?

By calculating the Pearson Correlation Coefficient, it is possible to find the correlation between the number of lines ending on a non-zero value and the evaluation results. Table 6.5 holds the correlation coefficient. The results reveal that even though there technically is a positive correlation, the relationship between the variables is weak. Therefore, no direct correlation is found.

Table 6.5.: Correlation between the number of non-zero predictions for the rhyme word and the survey scores.

	Number of non-zero predictions
Poeticness	0.1658
Grammaticality	0.3040
Meaningfulness	0.2674

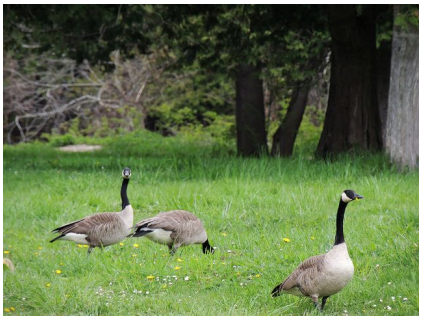
6.2.2. Human- and machine generated poetry experiment

The second experiment which was done was to see if the participants could tell the difference between the machine generated poetry and human written poetry. The experiment process was as follows: After 30 stanzas had been evaluated, the four highest rated poems were selected. The persons who had chosen the images was asked to write a poem with the same syllabic and rhyming constraints, i.e. each sentence has to have eight syllables and the two first and two last lines have to rhyme. This resulted in the experimental setup shown in Figure 6.2.:

It's been a long time and I've bees
But I don't want to get disease
I'm all out of love, oh so sweet
Just give me one more chance to meet

I am flying in the warm air
The other bees are very fair
It is a beautiful sunday
I am leaving later today

(a) Picture from "bee" (CC BY 2.0) by David Elliott.
URL: https://www.flickr.com/photos/drelliott0net/15105557167/.



I want to see swans in the sky
You and me, we are in for fly
If there's a lake out there for goose
There is no limit for excuse

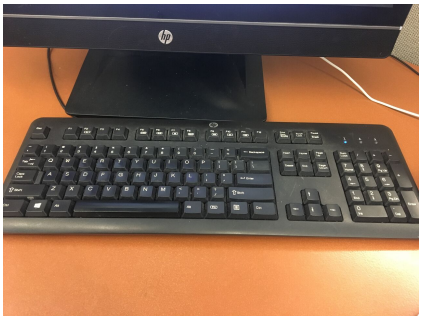
Eating dinner like a wild goose
Food is something that I can't lose
Although cooking is really tough
Nothing better can make me bluff

(b) Picture from "Canada Geese" (CC BY 2.0) by Kevin M Klerks.
URL: https://www.flickr.com/photos/ledicarus/34618458382/.

Broomstick on the back of my broom
Move on like dead man can assume
I don't know what to do and sweep
It makes me think that I'm asleep

Checking the status of my broom
I need the broom to sweep the room
I have no choice to keep waiting
It's hard for me to stop hating

(c) Picture from "broom" (CC BY 2.0) by danjo paluska.
URL: https://www.flickr.com/photos/sixmilliondollardan/3074916976/.



It's been a long time since the kick
It makes me want to hold you stick
And I know communication
I've got so much combination

Tapping on my keyboard at night
Hoping it will not cause a fight
You are my only listener
Don't let me be your prisoner

(a) Picture by Yumeng Sun with permission to use.

Figure 6.3.: The setup for Experiment 2.

For each of the images, the computer generated stanza and human evaluated stanza is shown in Figure 6.2. The top stanzas next to each image is the computer generated stanza, and the other stanza is the human written stanza. The question asked to the participants of the experiment is: *Which stanza is human generated?* 38 persons evaluated all four items, making it a total of 152 evaluations. The results are shown in Table 6.6. For simplicity, the computer generated stanzas are moved to the top next to each picture when visualized here, however, in the experiment, in Figure 6.2d the computer generated poetry was on the bottom.

Table 6.6.: This table shows the results of the second experiment.

Figure	Top stanza chosen	Bottom stanza chosen
Figure 6.2a	10	28
Figure 6.2b	5	33
Figure 6.2c	11	27
Figure 6.2d	9	29

In total 117 people chose the correct option, while 35 people chose the wrong option. This makes the participants have a 76.5% success ratio. Figure 6.2b was the easiest for the participants to figure out, where only five participants answered wrong.

6.2.3. Participant Sentiments and Qualitative Observations

The last part of the experiment was to get the participant sentiments. The question asked to the participants was the following: *After seeing the results of the images that you chose, and the poetry of the four images of experiment two, do you have any finishing thoughts?*. 29 of the 46 participants used the word *fun* when evaluating the poetry, while 20 participants used the word *random* when describing the poetry. Other notions such as *... interesting system ...* and *... the poems feel alive ...* were also common. A few sentiments used the word *bad*. Lastly, 7 participants mentioned *personification* as a *core value* of the generation system.

7. Evaluation and Conclusion

This chapter discusses the obtained results, and makes a conclusion in Section 7.3. Finally, suggestions for future work are presented.

7.1. Evaluation

The average score of 2.614 in the poeticness category tells that people overall found the stanzas in between *partially poetic* and *poetic* in terms of rhythm and rhyming. The median of poeticness is 2.8, and the standard deviation is 0.519. The standard deviation is quite big, representing a big spread of the sentiment of the participants. The average score of 2.381 in the grammar category indicates that the grammar is closer to being *partially grammatically correct* than being *grammatically correct*. The standard deviation of 0.470 indicates that the spread is smaller than the spread for poeticness, but still is quite a big spread. The average score of 2.050 in the meaningful tells that the system is on average evaluated as *partially meaningful*. The standard deviation of 0.712 also indicates that the results of meaningfulness are the most spread out, and therefore the most inconsistent feature of the system.

As for research question 1, there are several effects of using song lyrics as a data set for the LSTM network reflected in the poetry. These are:

- Extensive use of the first person pronoun, *I*.
- Repetition of several phrases.
- Extensive use of the word *love* and terms related to it.
- A big fraction of the vocabulary are predicted with a zero probability at any given point in time.

The extensive use of *I* is displayed in Table 7.1. The left column shows the number of times any form of *I*, e.g. *I've* and *I'll*, appears in the stanza. From the table it is clear that a big fraction of the stanzas include at least one form of *I*. The result of this can remind of the *personification* feature. *Personification* is when an object or abstract phenomenon is portrayed as

having human abilities and properties. For instance, the use of *I* in the stanza generated by a tripod as in Figure 6.1c makes the tripod feel, in the words of the participants, *alive*. It personifies the object.

Another point to consider when stating that the poetry generated is using personification is the sentiments of the participants in the experiments. As stated earlier, 7 participants mentioned the word *personification*, while others mentioned the object to become *alive*.

Table 7.1.: Percentage of poems containing *I*'s

# of *I*'s	% of poems that contains # of *I*'s
≥ 1	87%
≥ 2	42%
≥ 3	12%

Furthermore, the system often uses the same phrases in many poems. The four most common ones are:

- I don't know...
- I want to ...
- It's been a long time since...
- So come on...

Out of all the 153 poems generated 72% contains at least one, while 27% contains two or more of the phrases above. This is one of the implications of using song lyrics. The amount of diversity regarding lines in the stanzas is not very big, and therefore the system often reuses phrases in different stanzas.

Another fascinating point is the use of the word *love*. In data set 2, *love* appears 10,671 times, and is found in 7% of the songs. *Love* is found in 11% of the poems generated. It makes the poem love based. In addition, the four phrases described above fit nicely with the theme of *love*. Other phrases often used in conjunction with *love* are *The world is full of...* and *I can't live without...*.

To summarize, using song lyrics is interesting due to elements of personification emerging and the grammar in a line is usually good until the rhyme word appears, while it has trouble predicting said rhyme words. A point worth mentioning is that the song lyrics can be altered by, for instance,

removing songs containing a high percentage of *I*. This will lower the percentage of *I* present in the data set, however, it will not make rare words more common.

The system has a couple of limitations. The first one is Inception. If Inception classifies the input image in the wrong class, the system will perform poorly, because all related words and rhyme words will become related to the wrong class. Another limitation of this is that only the Top-1 class is used to find related words. The reason to only use the Top-1 result is because other results are often not related and might be wrong, therefore hurting the performance of the system. This means that only one object in the image is used. When inspection the architecture of the system, using object recognition instead of classification might yield better results, because of the ability to identify multiple objects in the image.

Another limitation is that the system only supports the use of one-word rhymes. For instance, choosing rhyme words such as *golden retriever* is not possible. *Retriever* will in this case become the rhyme word, and the rest of the line is generated without any notice of *golden*.

7.2. Discussion

Three different goals were introduced in Section 1.1. Whether these goals were reached or not, will be discussed in this section.

G1: Investigate computational poetry generation and image object recognition

Both poetry generation and image object recognition was investigated, with a main focus on the poetry generation. The field of poetry generation is thoroughly studied, however, some more investigation into deep learning and RNNs is possible to do. Regarding object recognition, only systems competing in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) were studied. It might have been a better option to study object recognition in general, and not only object classification.

G2: Develop a data set from scratch

A data set was gathered and developed from scratch. Several difficulties emerged, as explained in Chapter 5, but in the end more than 200,000 songs were gathered and pre-processed carefully. However, the data set is not flawless, and there is still content that is not English song lyrics present in the data set text. Upon closer inspection of the data set, some content that is

not song lyrics such as *+++The artist runs left+++* is still present, however, it does not have any noticeable impact on the system.

G3: Evaluate the results of a system that generates poetry from visual input

A design of an algorithm able to take an image as the input and output a stanza related to the image was made and implemented. Several interesting properties arose because of the data set the LSTM is trained on. Two evaluations have been conducted, namely experiment 2 and 3, to analyse the results of the system. The goal itself is reached, but more work could have been made towards evaluating if the participants found the system creative, and if so, in what ways they found it creative.

7.3. Conclusion

In this Thesis, a system capable of generating poetry of images was designed and implemented. The implementation was then used to generate 153 stanzas. The system includes a CNN for object classification, a module to find related words and rhyme pairs, and an LSTM network to score a tree search where nodes are representing the words being generated.

Three different experiments were conducted to evaluate the performance of the system. The first one looked at how training song lyrics on the LSTM network affected the word perplexity of different architectures. The second and third experiments were conducted with volunteering participants to evaluate the generated poetry.

The results of the system were varied, with a big standard deviation on the three criteria the system was evaluated on. The system was not able to consistently generate stanzas that are perceived by everyone to be aesthetically good. The best results were achieved when the LSTM network could predict the rhyme word with non zero prediction scores, however, only a weak correlation was found between the evaluation and the stanzas containing non zero prediction scores.

A data set was gathered and pre-processed to train the LSTM model. Various difficulties arose when gathering the data, the biggest being the variety of random content in the song lyrics. Upon closer inspection of the data set some content that is not song lyrics are still present, however, the fraction of this content does not have any noticeable impact on the system.

Overall the system will probably never write any poetic masterpiece, but the evaluations made by the participants indicate that some generated stan-

zas were subjectively enjoyable. This suggests that the implementation could be useful as a foundation for other poetry generation systems.

7.4. Possible future work

Several different features are possible to integrate into the system. This is because the system is big, and includes several different modules to generate the poetry. These modules can be changed, to see new interesting properties and features emerge from the poetry.

One option for trying to enhance the performance of the predictions is changing the LSTM network to a Sequence to Sequence Model (Cho et al., 2014). Several newer poetry generation systems such as Wang et al. (2016b) use this approach, and report good results.

Another possible change to the system is to train a separate model for fetching related words. One popular model for doing this is to train a word2vec model. Implementing this into the system could enhance performance by finding better related words and better rhyming words.

Training the LSTM network on poetry instead of song lyrics is also an interesting variation to test. Poetry has different properties than song lyrics, such as using a bigger variety of words more often. The evaluation of this Thesis' system indicates that training the LSTM network on another data set would yield very different results.

Lastly, the system can delve into various other fields and gather information from them. For instance, replacing the image recognition with sound recognition would make the system generate poetry given acoustic input.

Bibliography

Derek Attridge and Philip Davies Roberts. How poetry works: The elements of english poetry, 1987.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, volume 29, pages 65–72, 2005.

Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Jinho D Choi. Dynamic feature induction: The last gist to the state-of-the-art. In *Proceedings of NAACL-HLT*, pages 271–281, 2016.

Simon Colton, Jacob Goodwin, and Tony Veale. Full face poetry generation. In *Proceedings of the Third International Conference on Computational Creativity*, pages 95–102, 2012.

Amitava Das and Björn Gambäck. Poetic machine: Computational creativity for automatic poetry generation in bengali. In *5th International Conference on Computational Creativity, ICCC*, 2014.

Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

Belén Díaz-Agudo, Pablo Gervás, and Pedro A González-Calero. Poetry generation in colibri. In *European Conference on Case-Based Reasoning*, pages 73–87. Springer, 2002.

Pablo Gervás. Wasp: Evaluation of different strategies for the automatic generation of spanish verse. In *Proceedings of the AISB-00 symposium on creative & cultural aspects of AI*, pages 93–100, 2000.

Pablo Gervás. An expert system for the composition of formal spanish poetry. *Knowledge-Based Systems*, 14(3):181–188, 2001.

Pablo Gervás. Exploring quantitative evaluations of the creativity of automatic poets. In *Workshop on Creative Systems, Approaches to Creativity in Artificial Intelligence and Cognitive Science, 15th European Conference on Artificial Intelligence*, 2002.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191, 2016.

Jing He, Ming Zhou, and Long Jiang. Generating chinese classical poems with statistical machine translation models. In *AAAI*, 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, abs/1406.4729: 1904–1916, 2014. URL `http://arxiv.org/abs/1406.4729`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma thesis. Institut f. Informatik, Technische Univ. Munich. Advisor: J. Schmidhuber*, 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Alexey Grigorévich Ivakhnenko. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics*, 1(4):364–378, 1971.

Alexey Grigorévich Ivakhnenko and Valentin Grigorévich Lapa. Cybernetic predicting devices. Technical report, DTIC Document, 1966.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 2169–2178. IEEE, 2006.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE.*, volume 11, pages 2278–2324, 1998.

Samuel R Levin. *Linguistic structures in poetry.* Number 23. Mouton De Gruyter, 1962.

Robert P Levy. A computational model of poetic creativity with neural network as measure of adaptive fitness. In *Proceedings of the ICCBR-01 Workshop on Creative Systems*, 2001.

Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023*, 2016.

Hisar Manurung. An evolutionary algorithm approach to poetry generation. 2004.

Hisar Maruli Manurung. A chart generator for rhythm patterned text. In *Proceedings of the First International Workshop on Literature in Cognition and Computer*, pages 15–19, 1999.

Ruli Manurung, Graeme Ritchie, and Henry Thompson. Using genetic algorithms to create meaningful poetic text. *Journal of Experimental & Theoretical Artificial Intelligence*, 24(1):43–64, 2012.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4): 115–133, 1943.

Zbigniew Michalewicz. Genetic algorithms + data structures = evolution programs(2nd, extended ed.). *Springer-Verlag New York, Inc., New York, NY, USA*, 1994.

Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. Association for Computational Linguistics, 2004.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Yael Netzer, David Gabay, Yoav Goldberg, and Michael Elhadad. Gaiku: Generating haiku with word associations norms. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 32–39. Association for Computational Linguistics, 2009.

Hugo Oliveira. Automatic generation of poetry: an overview. *Universidade de Coimbra*, 2009.

Hugo Gonçalo Oliveira. Poetryme: a versatile platform for poetry generation. *Computational Creativity, Concept Invention, and General Intelligence*, 1: 21, 2012.

Hugo Gonçalo Oliveira, F Amılcar Cardoso, and Francisco C Pereira. Exploring different strategies for the automatic generation of song lyrics with tra-la-lyrics. In *Proceedings of 13th Portuguese Conference on Artificial Intelligence, EPIA*, pages 57–68, 2007.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia*, pages 311–318, 2002.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Robert Speer and Catherine Havasi. Representing general relational knowledge in conceptnet 5. *Proceedings of LREC*, 2012.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, pages 1929–1958, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

Jukka Toivanen, Hannu Toivonen, Alessandro Valitutti, Oskar Gross, et al. Corpus-based generation of content and form in poetry. In *Proceedings of the Third International Conference on Computational Creativity*, 2012.

Hannu Toivonen, Oskar Gross, Jukka M Toivanen, and Alessandro Valitutti. On creative uses of word associations. In *Synergies of Soft Computing and Statistics for Intelligent Data Analysis*, pages 17–24. Springer, 2013.

Qixin Wang, Tianyi Luo, and Dong Wang. Can machine generate traditional chinese poetry? a feigenbaum test. In *Advances in Brain Inspired Cognitive Systems: 8th International Conference, BICS 2016, Beijing, China, November 28-30, 2016, Proceedings 8*, pages 34–46. Springer, 2016a.

Zhe Wang, Wei He, Hua Wu, Haiyang Wu, Wei Li, Haifeng Wang, and Enhong Chen. Chinese poetry generation with planning based neural network. *arXiv preprint arXiv:1610.09889*, 2016b.

Paul John Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975.

Jane Shaw Whitfield. *Whifield's University Rhyming Dictionary*. Barnes and Noble Books, 1951.

Geraint A Wiggins. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7):449–458, 2006.

Rui Yan, Han Jiang, Mirella Lapata, Shou-De Lin, Xueqiang Lv, and Xiaoming Li. i, poet: Automatic chinese poetry composition through a generative summarization framework under constrained optimization. In *IJCAI*, 2013.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Xingxing Zhang and Mirella Lapata. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680, 2014.

Appendix

A. Various poems generated by the system

In this appendix, more generated stanzas are showcased. Figure 1 shows ten of some of the highest scoring stanzas, while Figure 2 shows ten of the poorest scoring stanzas.

URL to Creative Common Attribution 2.0 (CC BY 2.0):
https://creativecommons.org/licenses/by/2.0/legalcode

You don't know what to do and fling
Blow your mind in the back of swing
I will be there for a while sway
When it all comes down on my play

Score: [2.8, 2.7, 3.0]

(a) Picture from "swinging" (CC BY 2.0) by EvelynGiggles.
URL: https://www.flickr.com/photos/evelynishere/3871627155/.

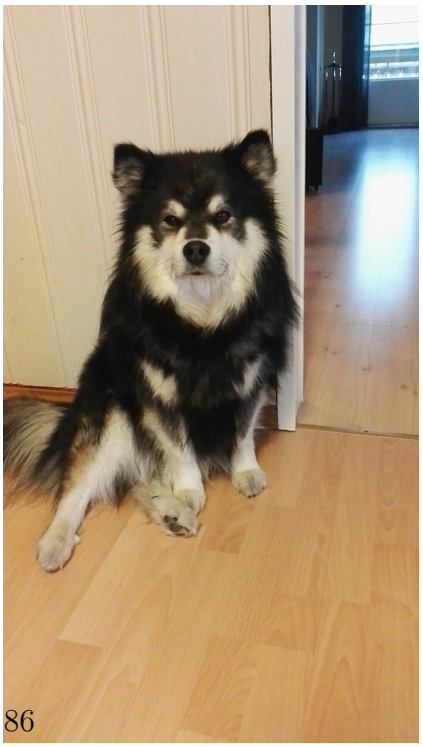
It's time to let it go fountain
Cheat a new world and the mountain
So come on love is in fountains
I don't know what to do mountains

(b) Picture by "Fountains @Bellagio" (CC by 2.0) by Sheila Thomson.
URL: https://www.flickr.com/photos/sheilaellen/5993290982/.

I can see it in her eyes clear
You don't have to do dish beer
The flask of love is like a pane
So come on and tell me that rain

Score: [2.8, 2.5, 2.4]

(c) Picture by Ole Steinar Skrede with permission to use.



I don't know what to say race
So come on take me home chase
All the things that we can do bark
I'll tell you where I've been in arc

(d) Picture by Ole Marius Husby Evensen with permission to use.

I want to be with you and shade
The time is right in front of aid
It's too late for me at all store
My hearts says so hats on the floor

Score: [3.0, 2.0, 2.5]

(e) Picture by Markus Andersson with permission to use.

The sun is in my big backyard
I don't know why you're a courtyard
Railing and row of bungalow
I've got a brand new patio

Score: [2.7, 2.7, 3.0]

(f) Picture by Yumeng Sun with permission to use. The image got classified as a patio, but still managed to get a high score.

I want to be with you in mug
So come on and take my hand jug
Let me get you out of here for pot
The coffee we make is a lot

Score: [2.8, 2.5, 3.0]

(g) Picture from "Cofee Pot" (CC BY 2.0) by Liam Dunn.
URL: https://www.flickr.com/photos/liamdunn/2664769983/.

I don't know what to do hammer
My head is full of love glamour
You make me feel this way chisel
It's like a hole in the drizzle

Score: [2.8, 2.7, 2.8]

(h) Picture from "Hammer" (CC BY 2.0) by James Bowe.
URL: https://www.flickr.com/photos/jamesrbowe/6371964415/.

I don't know what to do cut small
The world is full of love game ball
I'm just a fool in Ney York sport
I've got it all on my own court

Score: [2.8, 2.5, 2.2]

(i) Picture from "Tennis" (CC BY 2.0) by PughPugh.
URL: https://www.flickr.com/photos/hughgallagher/6041535945/.

I don't know what to do with wheat
My heart is full of love so sweet
You make me feel this way and crop
It feels like the first time that drop

(j) Picture from "Corn" (CC BY 2.0) by Ozzy Delaney.
URL: https://www.flickr.com/photos/24931020%40N02/15362528175/.

Figure A.1.: Ten of the highest rated poems.

I don't know what to do four drink
You can tell me I'm in love sink
The time is right for a long tub
My eyes are full of tears and scrub

Score: [2.1, 2.4, 1.5]

(a) Picture by "Toilet Paper" (CC by 2.0) by Rudolf Vicek.
URL: https://www.flickr.com/photos/kelehen/14482400697/.

I can't get skin out of my root
No need to run and hide dish fuit
So come on now let me grape sauce
Preserved in the arms like a toss

Score: [1.0, 1.0, 1.0]

(b) Picture by "Cucumber" (CC by 2.0) by Mani Bhaskar.
URL: https://www.flickr.com/photos/mani_bhaskar23/31929774703/.

I want to be the one that roar
You camel and make me adore
I'm just a jackal in canine
Black or white on my own airline

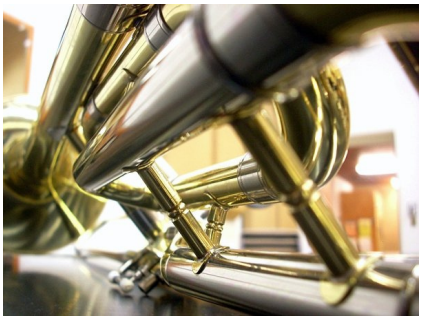
Score: [1.7, 2.4, 1.0]

(c) Picture from Ole Marius Husby Evensen with permission to use.

I don't know how to do leagues sports
My head is full of love and courts
So come on and let me take football
I'll give you one more protocol

Score: [1.5, 2.0, 1.0]

(d) Picture by "Student athletes" (CC by 2.0) by DVIDSHUB.
URL: https://www.flickr.com/photos/dvids/8347432644/.



I want to know swan baritone
In the lining of saxophone
I'm just a suite and no one flute
You don't sound me at all now suit

Score: [1.0, 1.0, 2.0]

(e) Picture by "Trombone" (CC by 2.0) by stephanie.lafayette.
URL: https://www.flickr.com/photos/93006384@N05/8468273380/.

I don't know what to do with blue
The sun is in my eyes and hue
So come on let me see skin bread
I'll show you how it should be red

Score: [3.0, 2.2, 1.0]

(f) Picture from Øyvind Nygård with permission to use.

I want to be the one that dish
A piece of my heart dollar fish
You make me feel this way seafood
All I've got is shot altitude

Score: [2.0, 2.3, 1.8]

(g) Picture from Christian Hundstad with permission to use.

Fox on the edge of my mind coop
I don't know what to do with soup
I'm just a man and not one hen
I've been down all over again

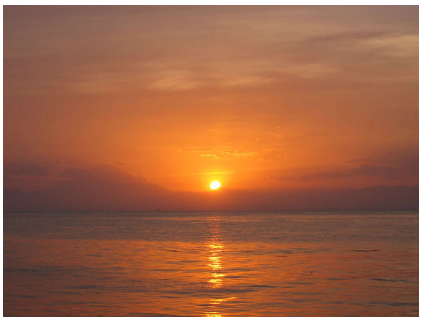
Score:[2.9, 2.7, 1.0]

(h) Picture from "Cock" (CC BY 2.0) by Tim Green.
URL: https://www.flickr.com/photos/atoach/13651523025/.

I don't know what matrimony
You can be the ceremony
But your love is just a game dress
It won't come to pass all in us

Score: [2.2, 2.3, 1.3]

(i) Picture by "1452077_638142362904714_2032839114_n" (CC by 2.0) by Shawn Whitman.
URL: https://www.flickr.com/photos/149105123@N05/34963239296/.



It's time to get out of here shore
Man on the floor but no one fore
It comes and goes salt like a gray
I don't know what life is for bay

Score: [2.8, 2.5, 1.5]

(j) Picture by "Sunset" (CC by 2.0) by skyseeker.
URL: https://www.flickr.com/photos/skyseeker/10494854/.

Figure A.2.: Ten of the lowest rated poems.